La clase Arrays

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**). Esta clase posee métodos muy interesantes para utilizar sobre arrays.
Su uso es
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 v[]=new int[10];
Arrays.fill(v,12);
```

También permite decidir desde que índice hasta qué índice rellenamos:

```
Arrays.fill(v,1,3,-1);
System.out.println("\nArray v relleno de 12, de la posición 1ª a la 2ª -1");
```

```
12  -1  -1  12  12  12  12  12  12  12
```

equals

Compara dos arrays y devuelve true si son iguales. Se consideran iguales si son del mismo tipo, tamaño y contienen los mismos valores.

```
int v[]=new int[10];
Arrays.fill(v,12);
int w[]=new int[10];
Arrays.fill(w,12);
Arrays.fill(w,1,3,-1);
System.out.println("\nArray w");
for (int i=0;i<w.length;i++)
    System.out.print(w[i]+"  ");

if (Arrays.equals(v,w)==true) System.out.println("\nSon arrays iguales");
else System.out.println("\nLos arrays NO son iguales");
```

sort

Permite ordenar un array en orden ascendente. Se pueden ordenar sólo una serie de elementos desde un determinado punto hasta un determinado punto.

```
int x[]={41,52,12,23,47,28,22,73,91,52};
Arrays.sort(x);
```

```
int y[]={41,51,12,9,7,8,22,73,91,52};
Arrays.sort(y,2,5);
System.out.println("\nArray y ordenado desde el índice 2 al índice 4");
```

```
41 51 7 9 12 8 22 73 91 52
```

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 o -1 si el elemento no está en el array. Ejemplo:

```
int x[]={41,52,12,23,47,28,22,73,91,52};
Arrays.sort(x);
int n=9;
int indice=Arrays.binarySearch(x,n);
System.out.println(n+((indice!=-1)?(" está en el índice "+indice):(" no está en el vector")));
n=28;
indice=Arrays.binarySearch(x,n);
System.out.println(n+((indice!=-1)?(" está en el índice "+indice):(" no está en el vector")));
```

9 no está en el vector
28 está en el índice 3

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 la 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[]={3,3,3,3,3,3,3,3,3};
System.arraycopy(uno, 0, dos, 0, uno.length);
for (int i=0;i<=8;i++){
       System.out.print(dos[i]+" ");
} //Sale 112333333
```

¿Por qué arraycopy?
Si tenemos:

```
int v[]={41,52,12,23,47,28,22,73,91,52};
System.out.println("Array v ");
for (int i=0;i<v.length;i++)
     System.out.print(v[i]+" ");
int w[];
w=v;
Arrays.sort(v);
System.out.println("\nArray v ");
for (int i=0;i<v.length;i++)
     System.out.print(v[i]+" ");
System.out.println("\nArray w ");
for (int i=0;i<w.length;i++)
     System.out.print(w[i]+" ");
 Arrays.fill(w,3,7,1);
 System.out.println("\nArray v ");
 for (int i=0;i<v.length;i++)
     System.out.print(v[i]+" ");
```

```
Array v
12 22 23 28 41 47 52 52 73 91
Array w
12 22 23 28 41 47 52 52 73 91
Array v
12 22 23 1 1 1 1 52 73 91
```

En w=v; quiere decir que w apunta a la dirección donde está guardado v, cuando algún valor de v cambia, también lo hace en w.
Si hacemos:

```
int v[]={41,52,12,23,47,28,22,73,91,52};
System.out.println("Array v ");
for (int i=0;i<v.length;i++)
     System.out.print(v[i]+" ");
int w[]=new int[10];
System.arraycopy(v,0, w,0,v.length);
Arrays.sort(v);
System.out.println("\nArray v ");
for (int i=0;i<v.length;i++)
     System.out.print(v[i]+" ");
System.out.println("\nArray w ");
for (int i=0;i<w.length;i++)
     System.out.print(w[i]+" ");
 Arrays.fill(w,3,7,1);
 System.out.println("\nArray v ");
 for (int i=0;i<v.length;i++)
     System.out.print(v[i]+" ");
```

```
Array v
41 52 12 23 47 28 22 73 91 52
Array v
12 22 23 28 41 47 52 52 73 91
Array w
41 52 12 23 47 28 22 73 91 52
Array v
12 22 23 28 41 47 52 52 73 91
```

clase String

Introducción

Para Java las cadenas de texto son objetos especiales. Los textos deben manejarse creando objetos de tipo String. Ejemplo:

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”;

También se pueden crear objetos String sin utilizar constantes entrecomilladas, usando otros constructores:

```
char[] palabra = {'P','a','l','a','b','r','a'};//Array de char
String cadena = new String(palabra);
```

Comparación entre objetos String

Los objetos **String** pueden compararse directamente con los operadores de comparación (==) o con el método equals. 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.**

```
String cadena1 = "alajuela", cadena2 = "Alajuela";
if (cadena1.equals(cadena2)){
        System.out.println("Son iguales");
else
        System.out.println("Son diferentes");
La impresión será: Son diferentes
```

 *cadena1.***equalsIgnoreCase**(cadena*2*). Como la anterior, pero en este caso no se tienen en cuenta mayúsculas y minúsculas.

```
String cadena1 = "alajuela";
String cadena2 = "Alajuela";
if (cadena1.equalsIgnoreCase(cadena2)){
        System.out.println("Son iguales");
else
        System.out.println("Son diferentes");
La impresión será: Son iguales
```

 *s1.***compareTo**(*s2*). Compara ambas cadenas, considerando el orden alfabético, devuelve un número.
Si la primera cadena es mayor en orden alfabético que la segunda devuelve >0
Si son iguales devuelve 0
Si es la primera cadena es menor a la segunda devuelve <0.
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*.

```
String cad1="1234";
String cad2="1334";
valor=cad1.compareTo(cad2);
if(valor==0)  System.out.println("Son iguales");
else if (valor<0) System.out.println(cad1+" es menor que "+ cad2);
        else System.out.println(cad1+" es may que "+ cad2);
```

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

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);
```

En el ejemplo se observa que este método pertenece a la clase String directamente, no hay que utilizar el nombre del objeto creado (como se verá más adelante, es un método estático).

Métodos de las variables de las cadenas

Son métodos que poseen las propias variables de cadena. Para utilizarlos basta con poner el nombre del 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 texto=”Prueba”;
System.out.println(texto.length());//Escribe 6
```

concatenar cadenas

Se puede hacer de dos formas, utilizando el método **concat** o con el operador +.
Ejemplo:

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

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, una excepción tipo **IndexOutOfBoundsException**. Ejemplo:

```
String s=”Prueba”;
char c=s.charAt(2); //c valdrá ‘u’
```

substring

Da como resultado una porción del texto de la cadena. La porción se toma desde una índice inicial hasta un índice final (sin incluir ese índice final). Si los índices indicados no son válidos ocurre una excepción de tipo **IndexOutOfBoundsException.** Ejemplo:

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

indexOf

Devuelve el primer índice en el 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”)); //Da 15
```

Se puede buscar desde una determinada posición. En el ejemplo anterior:
System.out.println(s1.indexOf("que",16)); *//Ahora da 26*

lastIndexOf

Devuelve el último índice en el 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"); *//Da 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"); *//Da 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** a un String para almacenar 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";
String s2=s1.replace("ar","er");*//Da Cazer ermadillos*
System.out.println(s2);*//Sigue valiendo Cazar armadillos*

toUpperCase

Devuelve la versión en mayúsculas de la cadena.

toLowerCase

Devuelve la versión en minúsculas de la cadena.

ToCharArray

Obtiene un array de caracteres a partir de una cadena.

Lista completa de métodos

char objetoString.charAt(int index**)** Proporciona el carácter que está en la posición dada por el entero *index.*
int objetoString.compareTo(string s**)** Compara las dos cadenas. Devuelve un valor menor que cero si la cadena *s* es mayor que la original, devuelve o si son iguales y devuelve un valor mayor que cero si *s* es menor que la original.
int objetoString.compareToIgnoreCase(string s) Compara dos cadenas, pero no tiene e cuenta si el texto es mayúsculas o no.
String objetoString.concat(String s**)** Añade la cadena *s* a la cadena original.
String objetoString.copyValueOf(char[] data**)** Produce un objeto **String** que es igual al array de caracteres *data*.
boolean objetoString.endsWith(String s**)** Devuelve **true** si la cadena termina con el texto *s*
boolean objetoString.equals(String s**)** Compara ambas cadenas, devuelve **true** si son iguales
boolean objetoString.equalsIgnoreCase(String s**)** Compara ambas cadenas sin tener en cuenta las mayúsculas y las minúsculas.
byte[] objetoString.getBytes() Devuelve un array de caracteres que toma a partir de la cadena de texto
void objetoString.getBytes(int srcBegin, **int** srcEnd, **char[]** dest, **int** dstBegin);
Almacena el contenido de la cadena en el array de caracteres *dest*. Toma los caracteres desde la posición *srcBegin* hasta la posición *srcEnd* y les copia en el array desde la posición *dstBegin*
int objetoString.indexOf(String s**)** Devuelve la posición en la cadena del texto s
int objetoStringindexOf(String s, **int** primeraPos) Devuelve la posición en la cadena del texto s, empezando a buscar desde la posición *PrimeraPos*
int objetoString.lastIndexOf(String s**)** Devuelve la última posición en la cadena del texto s
int objetoString.lastIndexOf(String s, **int** primeraPos) Devuelve la última posición en la cadena del texto s, empezando a buscar desde la posición *PrimeraPos*
int objetoString.length() Devuelve la longitud de la cadena
String objetoString.replace(char carAnterior, **char** ncarNuevo)
Devuelve una cadena idéntica al original pero que ha cambiando los caracteres iguales a *carAnterior* por *carNuevo*
String objetoString.replaceFirst(String str1, **String** str2**)** Cambia la primera aparición de la cadena str1 por la cadena str2
String objetoString.replaceFirst(String str1, **String** str2**)** Cambia la primera aparición de la cadena uno por la cadena dos
String objetoString.replaceAll(String str1, **String** str2**)** Cambia la todas las apariciones de la cadena uno por la cadena dos
String objetoString.startsWith(String s**)** Devuelve **true** si la cadena comienza con el texto *s.*
String objetoString.substring(int primeraPos, **int** segundaPos**)** Devuelve el texto que va desde *primeraPos* a *segunaPos*.
char[] objetoString.toCharArray() Devuelve un array de caracteres a partir de la cadena dada
String objetoString.toLowerCase() Convierte la cadena a minúsculas

String objetoString.toLowerCase(Locale local**)** Lo mismo pero siguiendo las instrucciones del argumento *local*
String objetoString.toUpperCase() Convierte la cadena a mayúsculas
String objetoString.toUpperCase(Locale local**)** Lo mismo pero siguiendo las instrucciones del argumento *local*
String objetoString.trim() Elimina los blancos que tenga la cadena tanto por delante como por detrás
static String valueOf(*tipo* elemento**)** Devuelve la cadena que representa el valor *elemento*. Si elemento es booleano, porejemplo devolvería una cadena con el valor **true** o **false**
Ejemplo,
String x=String.valueOf(897); // x="897"

Clases para tipos básicos

En Java se dice que todo es considerado un objeto. Para hacer que esta filosofía sea más real se han diseñado una serie de clases relacionadas con los tipos básicos, también llamadas clases wrappers o contenedoras. El nombre de
estas clases es:

al tipo básico..

clase	Representa al tipo…
java.lang.**Void**	**Void**
java.lang.**Boolean**	**Boolean**
java.lang.**Character**	**Char**
java.lang.**Byte**	**Byte**
java.lang.**Short**	**Short**
java.lang.**Integer**	**Int**
java.lang.**Long**	**Long**
java.lang.**Float**	**Float**
java.lang.**Double**	**Double**

Hay un método en cada una de esas clases que se llama **parse**.(analizar)
Ejemplo:

String s="2500";
int y=Integer.parseInt(s);
short z=Short.parseShort(s);
double c=Short.parseDouble(s);
byte x=Byte.parseByte(s); //aquí habría pérdida de información 2500 > 255.

También está el método en cada una de esas clases que se llama **toString** (convierte a String)
Ejemplo:

```
int x=89;
String cad1=Integer.toString(x);
double d=89.234567891234456d;
String cad2=Double.toString(d);
String cad3=Float.toString((float)d);
```

En la clase **Character** están los métodos:

Declaraciones

La primera sentencia creará una variable carácter y la segunda un objeto Character:

```
char c;
Character C;
```

Comprobaciones booleanas

```
Character.isLowerCase( c )
Character.isUpperCase( c )
Character.isDigit( c )
Character.isSpace( c )
Character.isLetter( c )
```

En este caso, si tuviésemos un objeto Character **C** , no se podría hacer *C.isLowerCase* , porque no se ha hecho un **new** de Character. Estas funciones son estáticas y no conocen al objeto, por eso hay que crearlo antes.

```
char c='o';
if (Character.isLetter(c)) System.out.println(c +" es una letra");
```

Traslaciones de caracteres

```
char c2 = Character.toLowerCase( c );
char c2 = Character.toUpperCase( c );
```

Traslaciones de carácter/dígito

```
int i = Character.digit( c,base ); //base es 8,10,16,si es octal,
decimal, hexadecimal
char c = Character.forDigit( i,base );
```

Métodos de la clase Character

```
Character C = new Character( 'J' );
char c = C.charValue();
String s = C.toString();
```

También están los valores Mínimo y Máximo:

Integer.MIN_VALUE	**-2^{31}**	**-2.147483648E9**
Integer.MAX_VALUE	**$2^{31}-1$**	
Byte.MIN_VALUE	**-2^{7}**	**-128**
Byte.MAX_VALUE	**$2^{7}-1$**	**127**
Short.MIN_VALUE	**-2^{15}**	**-32768**
Short.MAX_VALUE	**$2^{15}-1$**	**32767**
Long.MIN_VALUE	**-2^{63}**	
Long.MAX_VALUE	**$2^{63}-1$**	

Float.MIN_VALUE	2^{-149}	
Float.MAX_VALUE	$(2-2^{-23})\cdot 2^{127}$	
Float.INFINITY_NEGATIVE		
Float.INFINITY_POSITIVE		
Float.NaN	valor no válido	No un Número (Not a Number)
Double.MIN_VALUE		
Double.MAX_VALUE		
Double.INFINITY_NEGATIVE		
Double.INFINITY_POSITIVE		
Double.NaN		

Método isNaN(valor) es para comprobar si el valor no es válido (se ha realizado una división 0.0/0.0).