



**UNIVERSIDAD
METROPOLITANA DE
HONDURAS**
Innovación, Valores, Liderazgo

Funciones Matematicas y de texto

Catedratico Ing. Allan Lopez
Programacion Orientada a Objetos
Herman Eliezer Meza 201902468

A continuación, mostraremos las funciones más importantes :

Función matemática	Significado	Ejemplo de uso	Resultado
abs	Valor absoluto	<code>int x = Math.abs(2.3);</code>	<code>x = 2;</code>
atan	Arcotangente	<code>double x = Math.atan(1);</code>	<code>x = 0.78539816339744;</code>
sin	Seno	<code>double x = Math.sin(0.5);</code>	<code>x = 0.4794255386042;</code>
cos	Coseno	<code>double x = Math.cos(0.5);</code>	<code>x = 0.87758256189037;</code>
tan	Tangente	<code>double x = Math.tan(0.5);</code>	<code>x = 0.54630248984379;</code>
exp	Exponenciación neperiana	<code>double x = Math.exp(1);</code>	<code>x = 2.71828182845904;</code>
log	Logaritmo neperiano	<code>double x = Math.log(2.7172);</code>	<code>x = 0.99960193833500;</code>
pow	Potencia	<code>double x = Math.pow(2.3);</code>	<code>x = 8.0;</code>
round	Redondeo	<code>double x = Math.round(2.5);</code>	<code>x = 3;</code>
random	Número aleatorio	<code>double x = Math.random();</code>	<code>x = 0.20614522323378;</code>
floor	Redondeo al entero menor	<code>double x = Math.floor(2.5);</code>	<code>x = 2.0;</code>
ceil	Redondeo al entero mayor	<code>double x = Math.ceil(2.5);</code>	<code>x = 3.0;</code>

Ejemplos de Uso:

```
1int valor;  
2double res;  
3valor=10  
4res=Math.sqrt(valor);  
5System.out.println(res);
```

Math.sqrt(valor) Raiz cuadrada

```
double a,b,c,valor,rad;  
1valor=30;  
2//primero convertimos valor en radianes  
3rad=Math.PI/180 * valor;  
4a=Math.sin(rad);  
5b=Math.cos(rad);  
6c=Math.tan(rad);  
7System.out.println("sen("+valor+") es  
8"+a);  
9System.out.println("cos("+valor+") es  
1"+b);  
0System.out.println("tan("+valor+") es  
"+c);
```

Para trigonometria (sen, cos, tan)

```
1double a,b,c,pi,e,valor,rad;  
2//valor de pi=3.14...  
3pi=Math.PI;  
4//valor de e  
5e=Math.E;  
6//logaritmo neperiano Ln()  
7a=Math.log(e);  
8//logaritmo en base 10()  
9b=Math.log10(10);
```

Logaritmos (ln, log)

Math.random() Numeros Aleatorios:

Cuando se utiliza esta función el resultado obtenido esta entre 0 y 1, es decir 0.98 0.01 etc.

```
1double z=Math.random();
2System.out.println(z);
```

Si necesitamos obtener numero aleatorio entre 1 y 100 se hace lo siguiente

```
1double
2z=Math.random()*100;
3System.out.println(z);
```

Si necesitamos obtener numero aleatorio entre 5 y 20 se hace lo siguiente

```
1int z=(int)
2(Math.random()*15)+5;
3System.out.println(z)
```

Limitar numero de decimales:

Ahora bien si necesitamos mostramos el resultado de una raiz cuadrada por ejemplo de 5, Math.sqrt(5) =2.23606797749979, pero quisieramos que nos muestre con dos decimales es decir 2.24 hacemos lo siguiente::

```
1import java.text.DecimalFormat;
2
3public class Matematicas {
4
5    public static void main(String[] args) {
6        double x=Math.sqrt(5);
7        DecimalFormat d = new
8        DecimalFormat("0.00");
9        System.out.println(d.format(x));
10    }
11}
```




Funciones de Texto

Funciones con Cadenas

Una vez que hemos visto lo sencillo que es crear una cadena de texto vamos a echar un vistazo a los métodos que nos permiten manipular la cadena de texto. Si tuviésemos que ordenar dichos métodos podríamos llegar a la siguiente división:

- Información básica de la cadena

 - Búsqueda de caracteres


 - Comparación de Cadenas

 - Búsqueda de subcadenas

 - Manejo de subcadenas

 - Manejo de caracteres

- Conversión a String: `valueOf()`



Información básica de la cadena

.length() Nos devuelve el tamaño que tiene la cadena.

char charAt(int index)

Devuelve el carácter indicado como índice. El primer carácter de la cadena será el del índice 0. Junto con el método **.length()** podemos recuperar todos los caracteres de la cadena de texto. Hay que tener cuidado. Ya que si intentamos acceder a un índice de carácter que no existe nos devolverá una excepción **IndexOutOfBoundsException**.

Búsqueda de caracteres

Tenemos un conjunto de métodos que nos permiten buscar caracteres dentro de cadenas de texto. Y es que no nos debemos de olvidar que la cadena de caracteres no es más que eso: una suma de caracteres.

int indexOf(int ch) Nos devuelve la posición de un carácter dentro de la cadena de texto. En el caso de que el carácter buscado no exista nos devolverá un -1. Si lo encuentra nos devuelve un número entero con la posición que ocupa en la cadena.

int indexOf(int ch, int fromIndex) Realiza la misma operación que el anterior método, pero en vez de hacerlo a lo largo de toda la cadena lo hace desde el índice (fromIndex) que le indiquemos.

int lastIndexOf(int ch) Nos indica cual es la última posición que ocupa un carácter dentro de una cadena. Si el carácter no está en la cadena devuelve un

int lastIndexOf(int ch, int fromIndex) Lo mismo que el anterior, pero a partir de una posición indicada como argumento.



Comparación de Cadenas

Los métodos de comparación nos sirven para comparar si dos cadenas de texto son iguales o no. Dentro de los métodos de comparación tenemos los siguientes:

[boolean equals\(Object anObject\)](#) Nos permite comparar si dos cadenas de texto son iguales. En el caso de que sean iguales devolverá como valor "true". En caso contrario devolverá "false". Este método tiene en cuenta si los caracteres van en mayúsculas o en minúsculas. Si queremos omitir esta validación tenemos dos opciones. La primera es convertir las cadenas a mayúsculas o minúsculas con los métodos [toUpperCase\(\)](#) y [toLowerCase\(\)](#) respectivamente. Métodos que veremos más adelante. La segunda opción es utilizar el método [equalsIgnoreCase\(\)](#) que omite si el carácter está en mayúsculas o en minúsculas.

[boolean equalsIgnoreCase\(String anotherString\)](#) Compara dos cadenas de caracteres omitiendo si los caracteres están en mayúsculas o en minúsculas.

[int compareTo\(String anotherString\)](#) Este método es un poco más avanzado que el anterior, el cual, solo nos indicaba si las cadenas eran iguales o diferentes. En este caso compara a las cadenas léxicamente. Para ello se basa en el valor Unicode de los caracteres. Se devuelve un entero menor de 0 si la cadena sobre la que se parte es léxicamente menor que la cadena pasada como argumento. Si las dos cadenas son iguales léxicamente se devuelve un 0. Si la cadena es mayor que la pasada como argumento se devuelve un número entero positivo. Pero que es esto de "mayor, menor o igual léxicamente". Para describirlo lo veremos con un pequeño ejemplo.

Ejemplos:

```
s1 = "Cuervo"  
s2 = "Cuenca"  
s1.compareTo(s2);
```

Compararíamos las dos cadenas. Los tres primeros caracteres son iguales "Cue". Cuando el método llega al 4 carácter tiene que validar entre la r minúscula y la n minúscula. Si utiliza el código Unicode llegará a la siguiente conclusión.

```
r (114) > n(110)
```

Y nos devolverá la resta de sus valores. En este caso un 4. Hay que tener cuidado, porque este método no tiene en cuenta las mayúsculas y minúsculas. Y dichos caracteres, aún siendo iguales, tienen diferentes código. Veamos la siguiente comparación

```
s1 = "CueRvo"  
s2 = "Cuervo"  
s1.compareTo(s2);
```

Nuevamente los tres caracteres iniciales son iguales. Pero el cuarto es distinto. Por un lado tenemos la r minúscula y por otro la r mayúscula. Así:

```
R(82) < r(114)
```

¿Qué entero nos devolverá el método compareTo? ¿-32?

int compareToIgnoreCase(String str) Este método se comportará igual que el anterior. Pero ignorando las mayúsculas. Todo un alivio por si se nos escapa algún carácter en mayúsculas ;-) Otros métodos para la comparación de cadenas son:

```
boolean regionMatch( int thisoffset,String s2,int s2offset,int len );  
boolean regionMatch( boolean ignoreCase,int thisoffset,String s2, int s2offset,int 1 );
```




Búsqueda de Subcadenas

Este conjunto de métodos son, probablemente, los más utilizados para el manejo de cadenas de caracteres. Ya que nos permiten buscar cadenas dentro de cadenas, así como saber la posición donde se encuentran en la cadena origen para poder acceder a la subcadena. Dentro de este conjunto encontramos:

[int indexOf\(String str\)](#) Busca una cadena dentro de la cadena origen. Devuelve un entero con el índice a partir del cual está la cadena localizada. Si no encuentra la cadena devuelve un -1.

[int indexOf\(String str, int fromIndex\)](#) Misma funcionalidad que [indexOf\(String str\)](#), pero a partir de un índice indicado como argumento del método.

[int lastIndexOf\(String str\)](#) Si la cadena que buscamos se repite varias veces en la cadena origen podemos utilizar este método que nos indicará el índice donde empieza la última repetición de la cadena buscada.

[lastIndexOf\(String str, int fromIndex\)](#) Lo mismo que el anterior, pero a partir de un índice pasado como argumento.

[boolean startsWith\(String prefix\)](#) Probablemente mucha gente se haya encontrado con este problema. El de saber si una cadena de texto empieza con un texto específico. La verdad es que este método podía obviarse y utilizarse el [indexOf\(\)](#), con el cual, en el caso de que nos devolviese un 0, sabríamos que es el inicio de la cadena.

[boolean startsWith\(String prefix, int toffset\)](#) Más elaborado que el anterior, y quizás, y a mi entender con un poco menos de significado que el anterior.

[boolean endsWith\(String suffix\)](#) Y si alguien se ha visto con la necesidad de saber si una cadena empieza por un determinado texto, no va a ser menos el que se haya preguntado si la cadena de texto acaba con otra. De igual manera que sucedía con el método [startsWith\(\)](#) podríamos utilizar una mezcla entre los métodos [indexOf\(\)](#) y [length\(\)](#) para reproducir el comportamiento de [endsWith\(\)](#). Pero las cosas, cuanto más sencillas, doblemente mejores



Métodos con subcadenas

Ahora que sabemos como localizar una cadena dentro de otra seguro que nos acucia la necesidad de saber como substraerla de donde está. Si es que no nos podemos estar quietos...

[String substring\(int beginIndex\)](#) Este método nos devolverá la cadena que se encuentra entre el índice pasado como argumento (beginIndex) hasta el final de la cadena origen. Así, si tenemos la siguiente

```
String s = "Víctor Cuervo";
```

El método...

```
s.substring(7);
```

Nos devolverá "Cuervo".

[String substring\(int beginIndex, int endIndex\)](#) Si se da el caso que la cadena que queramos recuperar no llega hasta el final de la cadena origen, que será lo normal, podemos utilizar este método indicando el índice inicial y final del cual queremos obtener la cadena. Así, si partimos de la cadena...

```
String s = "En un lugar de la Mancha...";
```

El método...

```
s.substring(6,11);
```

Nos devolverá la palabra "lugar".



Manejo de caracteres

Otro conjunto de métodos que nos permite jugar con los caracteres de la cadena de texto. Para ponerles en mayúsculas, minúsculas, quitarles los espacios en blanco, reemplazar caracteres,....

[String toLowerCase\(\);](#)

Convierte todos los caracteres en minúsculas.

[String toUpperCase\(\);](#)

Convierte todos los caracteres a mayúsculas.

[String trim\(\);](#) Elimina los espacios en blanco de la cadena.

[String replace\(char oldChar, char newChar\)](#) Este método lo utilizaremos cuando lo que queramos hacer sea el reemplazar un carácter por otro. Se reemplazarán todos los

Conversión a String: valueOf()

Un potente conjunto de métodos de la clase [String](#) nos permite convertir a cadena cualquier tipo de dato básico: int, float, double,... Esto es especialmente útil cuando hablamos de números. Ya que en múltiples ocasiones querremos mostrarles como cadenas de texto y no en su representación normal de número. Así podemos utilizar los siguientes métodos:

[String valueOf\(boolean b\);](#)

[String valueOf\(int i\);](#)

[String valueOf\(long l\);](#)

[String valueOf\(float f\);](#)

[String valueOf\(double d\);](#)

[String valueOf\(Object obj\);](#)



String y StringBuffer

El paquete `java.lang` contiene dos clases de cadenas: `String` y `StringBuffer`. La clase `String` se utiliza cuando se trabaja con cadenas que no pueden cambiar. Por otro lado, `StringBuffer`, se utiliza cuando se quiere manipular el contenido de una cadena. El entorno de desarrollo Java proporciona dos clases para manipular y almacenar datos del tipo carácter: `String`, para cadenas constantes, y `StringBuffer`, para cadenas que pueden cambiar.

Como son constantes, los `Strings` son más económicos (utilizan menos memoria) que los `StringBuffers` y pueden ser compartidos. Por eso es importante utilizar `String` siempre que sea apropiado.

Conversión de objetos a Strings

toString() A veces es conveniente o necesario convertir un objeto a una cadena o `String` porque se necesitará pasarlo a un método que sólo acepta `Strings`. Por ejemplo, `System.out.println()` no acepta `StringBuffers`, por lo que necesita convertir el `StringBuffer` a `String` para poder imprimirlo. El método `reverse()` utiliza el método `toString()` de `StringBuffer` para convertirlo en un `String` antes de retornar. **return dest.toString();**