LA CLASE Math

La clase **Math** es una librería de métodos matemáticos incluida en el API de Java. **No es necesario importar** la clase **Math** para utilizar sus métodos, puesto que es parte del paquete **java.lang**, que se importa automáticamente por el compilador en todo proyecto Java.

La clase **Math** es ***public*** para que sus métodos puedan ser llamados desde cualquier parte de un proyecto Java que la importe, y es ***static,*** lo que indica que es una clase "helper" de utilidad cuyos métodos pueden ser invocados sin necesidad de instanciar objetos. Hablaremos en las Unidades 3 y 4 de los tipos de clases en Java. **Math** no es una clase "molde" para instanciar objetos, sino una clase "helper" para utilizar sus métodos (utilidades) directamente. Los **métodos** de la clase **Math** nos permitirán realizar fácilmente en Java operaciones matemáticas comunes.

Algunos Métodos de la clase Math

Math.abs(x)
Math.sin(double)
Math.cos(double)
Math.tan(double)
Math.asin(double)
Math.acos(double)
Math.atan(double)
Math.exp(double)
Math.log(double)
Math.sqrt(double)
Math.ceil(double)
Math.floor(double)
Math.rint(double)
Math.pow(a,b)
Math.round(x)
Math.random()
Math.max(a,b)
Math.min(a,b)
Math.E
Math.PI

Ejemplos

- Para elevar un número (x) a una potencia (y), se emplea **Math.pow(x, y)**
- Para hallar la raíz cuadrada de un número (x), se emplea **Math.sqrt(x)**
- Para redondear un número (x) con decimales al **int** más próximo se emplea **Math.round(x)**
- Para truncar un número (x) con decimales al **int** más próximo se emplea **Math.floor(x)**
- Para calcular el mayor de 2 números (x,y) se emplea **Math.max(x,y)**
- Para calcular el menor de 2 números (x,y) se emplea **Math.min(x,y)**
- Para obtener números aleatorios se emplea **Math.random()** *(Se verá más adelante en detalle)*

STRINGS – Cadenas de caracteres.

String es una **clase Java** que representa una **cadena de caracteres no modificable** (inmutable). No es necesario hacer ningún **import** para usarla, pues al igual que la clase **Math** está incluida en el paquete **java.lang,** que puede ser usado sin **importar** en nuestros proyectos **Java**.

Un **String** es cualquier cadena de caracteres. En Java los escribimos entre comillas dobles: *"cualquier texto"*

La clase **String** tiene un comportamiento dual, pues podemos crear Strings como cualquier otro **objeto** en el área de memoria **HEAP**, pero también podemos gestionarlos más eficazmente con un "pool" de Strings llamado **String Constant Pool,** que simplifica la gestión de Strings cuyo contenido se repite.

Por es posible declarar Strings de **2 formas:**

1. **String frase = new String("Hola a tod@s");**

//se crea en el área de memoria HEAP – dónde viven los objetos en Java

Formas de declarar, instanciar y asignar un String **como objeto**	
String frase = new String("Hola a tod@s");	Declaración + instanciación + asignación
String frase; frase = new String("Hola a tod@s"); String frase = new String(); frase= "Hola a tod@s";	Declaración Instanciación + asignación Declaración + instanciación Asignación
String frase; frase = new String(); frase= "Hola a tod@s";	Declaración Instanciación Asignación

2. **String frase = "Hola a todos";**

//Se crea en el HEAP, pero lo gestiona la String Constant Pool

Formas de declarar y asignar un String **en la String Constant Pool**	
String frase = "Hola a tod@s";	Declaración + asignación
String frase; frase = "Hola a tod@s";	Declaración Asignación

Para los/as principiantes, es más sencilla y recomendable la **2ª forma** si vamos a utilizar el **tipo String original,** pues la String Constant Pool hace una gestión más eficiente de los Strings.

Sin embargo, en el futuro seguramente nos veremos en la necesidad de usar tipos **String** más evolucionados y mutables(modificables), como las clases **StringBuilder** y **StringBuffer** (también incluidas en java.lang). En ese caso tendremos que declarar y manejar los Strings siempre como **objetos**, instanciándolos con **new StringBuilder()** o **new StringBuffer()**, tal y como hemos hecho de la **1ª forma.**

MÉTODOS DE LA CLASE STRING

La clase **String** proporciona métodos para el tratamiento de las cadenas de caracteres: conocer la longitud de una cadena, comparar cadenas, acceder a caracteres concretos, buscar y extraer una subcadena, copiar cadenas, convertir a mayúsculas o minúsculas, etc.

(operador concatenación – NO RECOMENDABLE). Además de los métodos de la clase String, existe una sobrecarga del operador **+** que sirve para "unir" 2 o más cadenas de caracteres. **Ejemplo**:

String frase = "hola" + "a" + "tod@s"; // El resultado de la concatenación "hola a tod@s"

En nuestros primeros ejemplos con java haremos uso de él por su simplicidad, pero a medio plazo **mejor obviarlo**, pues si lo usamos de forma repetida es muy ineficiente**,** y crea multitud de objetos auxiliares (2 cada vez que se usa) que pueden llegar a saturar la memoria HEAP.

Es recomendable descartar su uso y emplear el método **concat**: String frase= hola.**concat**("a todos");

Concat también tiene limitaciones por el carácter inmutable de los strings, pero **SIEMPRE** será mejor que el operador **+**, a la espera de que en un futuro usemos clases de tipo String más evolucionadas como **StringBuilder** y sus métodos **append(), insert(), delete(), replace() etc**

MÉTODO	DESCRIPCIÓN
concat(OtroString)	Concatena (une) el String original con "otroString"
length()	Devuelve la longitud de la cadena
indexOf('caracter')	Devuelve la posición de la primera aparición de *carácter* dentro del String. Devuelve -1 si no lo encuentra.
lastIndexOf('caracter')	Devuelve la posición de la última aparición de *carácter* dentro del String. Devuelve -1 si no lo encuentra.
charAt(n)	Devuelve el carácter que está en la posición n
substring(n1,n2)	Devuelve subcadena desde la posición n1 hasta n2 - 1
toUpperCase()	Devuelve la cadena convertida a mayúsculas
toLowerCase()	Devuelve la cadena convertida a minúsculas
equals(otroString)	Compara dos cadenas y devuelve true si son iguales
equalsIgnoreCase(otroString)	Igual que equals pero sin considerar mayúsculas y minúsculas
compareTo(OtroString)	Devuelve 0 si las dos cadenas son iguales. <0 si la primera es alfabéticamente menor que la segunda ó >0 si la primera es alfabéticamente mayor que la segunda.
compareToIgnoreCase(OtroString)	Igual que compareTo pero sin considerar mayúsculas y minúsculas.
valueOf(N)	Convierte el valor N a String. N puede ser de cualquier tipo.

Se pueden consultar todos estos métodos y constructores en el API de Java
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html

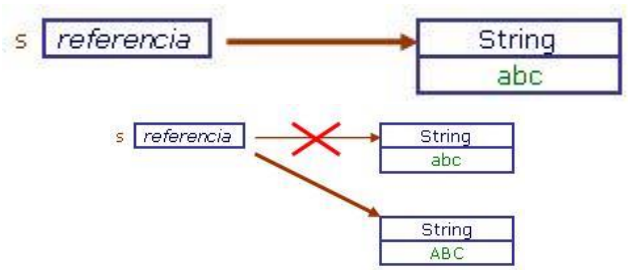
Ejemplos:

“Miguel”.concat(“Rodríguez”)	Miguel Rodríguez
“Miguel”.length()	6
“Miguel”.equals(“Miguel”)	true
“Miguel”.equals(“miguel”)	false
“Miguel”.equalsIgnoreCase(“miguel”)	true
“Miguel”.compareTo(“Saturnino”)	-6 (cualquier entero < 0)
“Miguel”.compareTo(“Miguel”)	0
“Miguel”.compareTo(“Michelin”)	4 (cualquier entero > 0)
“Miguel”.charAt(1)	'i'
“Miguel”.charAt(4)	'e'
“Miguel”.toCharArray()	{ 'M', 'i', 'g', 'u', 'e', 'l' }
“Miguel”.substring(1, 4)	“igu”
“Miguel.substring(1)	“iguel”
“tragaldabas”.indexOf('a')	2
“tragaldabas”.lasIndexOf('a')	9
“tragaldabas”.startsWith(“tragón”)	false
“tragaldabas”.endsWith(“dabas”)	true
“tragaldabas”.split(“a”)	{ “tr”, “g”, “ld”, “b”, “s” }

Los objetos String son inmutables (no es modificable el String original).

Los métodos que actúan sobre un String con la intención de modificarlo lo que hacen es crear un nuevo String a partir del original y devolverlo modificado.

Por ejemplo: La operación de convertir a mayúsculas o minúsculas un String no lo modificará, sino que creará y devolverá un nuevo String con el resultado de la operación**. String s = "abc"; s = s.toUpperCase();**



El **recolector de basura** de JAVA es el encargado de eliminar de forma automática los objetos a los que ya no hace referencia ninguna variable.