***LA CLASE Math***

La clase Math representa la librería matemática de Java. Las funciones que contiene son las de todos los lenguajes, parece que se han metido en una clase solamente a propósito de agrupación, por eso se encapsulan en Math, y lo mismo sucede con las demás clases que corresponden a objetos que tienen un tipo equivalente (Character, Float, etc.). El constructor de la clase es privado, por los que no se pueden crear instancias de la clase. Sin embargo, Math es *public* para que se pueda llamar desde cualquier sitio y *static* para que no haya que inicializarla.

***Funciones matemáticas***

Si se importa la clase, se tiene acceso al conjunto de funciones matemáticas estándar:

Math.abs( x )          para int, long, float y double  
Math.sin( double )  
Math.cos( double )  
Math.tan( double )  
Math.asin( double )  
Math.acos( double )  
Math.atan( double )  
Math.atan2( double,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 )       para double y float  
Math.random()         devuelve un double.

* **Math.random()** devuelve un número aleatorio entre 0.0 y 1.0, excluido este último valor, es decir, puede devolver 0.346442, 0.2344234, 0.98345,..

public class PruebaApp {

    public static void main(String[] args) {

         int num1=50;

        int num2=120;

         System.out.println("Números generados entre 0 y 20, con decimales (sin incluir el 0 y el 20)");

        for (int i=0;i<1000;i++){

            double numAleatorio=Math.random()\*20;

            System.out.println(numAleatorio);

        }

        System.out.println("Números generados entre 5 y 20, con decimales (sin incluir el 5 y el 20)");

        for (int i=0;i<1000;i++){

            double numAleatorio=Math.random()\*(20-5)+5;

            System.out.println(numAleatorio);

        }

        System.out.println("Números generados entre 50 y 120, sin decimales (sin incluir el 50 y el 120)");

        for (int i=0;i<1000;i++){

            int numAleatorio=(int)Math.floor(Math.random()\*(num1-num2)+num2);

            System.out.println(numAleatorio);

        }

                System.out.println("Números generados entre 50 y 120, sin decimales (incluyendo el 50 y el 120)");

        for (int i=0;i<1000;i++){

            int numAleatorio=(int)Math.floor(Math.random()\*(num1-(num2+1))+(num2));

            System.out.println(numAleatorio);

        }

    }

}

Math.max( a,b )       para int, long, float y double  
Math.min( a,b )       para int, long, float y double  
Math.E                para la base exponencial  
Math.PI               para PI

He aquí un ejemplo, [Mates.java](http://proton.ucting.udg.mx/tutorial/java/Fuentes/Cap3/Mates.java), de uso de algunas funciones de la clase Math:

class Mates {

public static void main( String args[] ) {

int x;

double rand,y,z;

float max;

rand = Math.random();

x = Math.abs( -123 );

y = Math.round( 123.567 );

z = Math.pow( 2,4 );

max = Math.max( (float)1e10,(float)3e9 );

System.out.println( rand );

System.out.println( x );

System.out.println( y );

System.out.println( z );

System.out.println( max );

}

}

***LA CLASE Character***

Al trabajar con caracteres se necesitan muchas funciones de comprobación y traslación. Estas funciones están empleadas en la clase **Character**. De esta clase sí que se pueden crear instancias, al contrario que sucede con la clase **Math**.

***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 )

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.

***Traslaciones de caracteres***

char c2 = Character.toLowerCase( c );

char c2 = Character.toUpperCase( c );

***Traslaciones de carácter/dígito***

int i = Character.digit( c,base );

char c = Character.forDigit( i,base );

***Métodos de la clase Character***

C = new Character( 'J' );

char c = C.charValue();

String s = C.toString();

***LA CLASE Float***

Cada tipo numérico tiene su propia clase de objetos. Así el tipo *float* tiene el objeto *Float*. De la misma forma que con la clase **Character**, se han codificado muchas funciones útiles dentro de los métodos de la clase **Float**.

***Declaraciones***

La primera sentencia creará una variable float y la segunda un objeto Float:

float f;

Float F;

***Valores de Float***

Float.POSITIVE\_INFINITY

Float.NEGATIVE\_INFINITY

Float.NaN

Float.MAX\_VALUE

Float.MIN\_VALUE

***Conversiones de Clase/Cadena***

String s = Float.toString( f );

f = Float.valueOf( "3.14" );

***Comprobaciones***

boolean b = Float.isNaN( f );

boolean b = Float.isInfinite( f );

La función *isNaN()* comprueba si f es un *No-Número*. Un ejemplo de no-número es raiz cuadrada de -2.

***Conversiones de Objetos***

Float F = new Float( Float.PI );

String s = F.toString();

int i = F.intValue();

long l = F.longValue();

float F = F.floatValue();

double d = F.doubleValue();

***Otros Métodos***

int i = F.hashCode();

boolean b = F.equals( Object obj );

int i = Float.floatToIntBits( f );

float f = Float.intBitsToFloat( i );

***LA CLASE Double***

Cada tipo numérico tiene su propia clase de objetos. Así el tipo *double* tiene el objeto *Double*. De la misma forma que con la clase **Character**, se han codificado muchas funciones útiles dentro de los métodos de la clase **Double**.

***Declaraciones***

La primera sentencia creará una variable double y la segunda un objeto Double:

double d;

Double D;

***Valores de Double***

Double.POSITIVE\_INFINITY

Double.NEGATIVE\_INFINITY

Double.NaN

Double.MAX\_VALUE

Double.MIN\_VALUE

***Métodos de Double***

D.isNaN();

Double.isNaN( d );

D.isInfinite();

Double.isInfinite( d );

boolean D.equals();

String D.toString();

int D.intValue();

long D.longValue();

float D.floatValue();

double D.doubleValue();

int i = D.hashCode();

Double V.valueOf( String s );

long l = Double.doubleToLongBits( d );

double d = Double.longBitsToDouble( l );

***LA CLASE Integer***

Cada tipo numérico tiene su propia clase de objetos. Así el tipo *int* tiene el objeto *Integer*. De la misma forma que con la clase **Character**, se han codificado muchas funciones útiles dentro de los métodos de la clase **Integer**.

***Declaraciones***

La primera sentencia creará una variable int y la segunda un objeto Integer:

int i;

Integer I;

***Valores de Integer***

Integer.MIN\_VALUE;

Integer.MAX\_VALUE;

***Métodos de Integer***

String Integer.toString( int i,int base );

String Integer.toString( int i );

int Integer.parseInt( String s,int base );

int Integer.parseInt( String s );

Integer Integer.valueOf( int i);

Integer Integer.valueOf( String s, int base );

Integer Integer.valueOf( String s );

int I.intValue();

long I.longValue();

float I.floatValue();

double I.doubleValue();

String I.toString();

boolean I.equals( Object obj );

public class Test{

public static void main(String args[]){

int x =Integer.parseInt("9");

double c = Double.parseDouble("5");

int b = Integer.parseInt("444",16);

System.out.println(x);

System.out.println(c);

System.out.println(b);

}

}

This produces the following result:

9

5.0

1092

***LA CLASE Long***

Cada tipo numérico tiene su propia clase de objetos. Así el tipo *long* tiene el objeto *Long*. De la misma forma que con la clase **Character**, se han codificado muchas funciones útiles dentro de los métodos de la clase **Long**.

***Declaraciones***

La primera sentencia creará una variable long y la segunda un objeto Long:

long l;

Long L;

***Valores de Long***

Long.MIN\_VALUE;

Long.MAX\_VALUE;

***Métodos de Long***

String Long.toString( long l,int base );

String Long.toString( long l );

long L.parseLong( String s,int base );

long L.parseLong( String s );

Long Long.valueOf( String s,int base );

Long Long.valueOf( String s );

int L.intValue();

long L.longValue();

float L.floatValue();

double L.doubleValue();

String L.toString();

int L.hashCode();

boolean L.equals( Object obj );

En los métodos *toString()*, *parseInt()* y *valueOf()* que no se especifica la **base** sobre la que se trabaja, se asume que es **base 10**.

***LA CLASE Boolean***

Los valores *boolean* también tienen su tipo asociado *Boolean*, aunque en este caso hay menos métodos implementados que para el resto de las clases numéricas.

***Declaraciones***

La primera sentencia creará una variable boolean y la segunda un objeto Boolean:

boolean b;

Boolean B;

***Valores de Boolean***

Boolean.TRUE;

Boolean.FALSE;

***Métodos de Boolean***

boolean B.booleanValue();

String B.toString();

boolean B.equals( Object obj );

## *LA CLASE String*

Java posee gran capacidad para el manejo de cadenas dentro de sus clases **String** y **StringBuffer**. Un objeto *String* representa una cadena alfanumérica de un valor constante que no puede ser cambiada después de haber sido creada. Un objeto *StringBuffer* representa una cadena cuyo tamaño puede variar.

Los Strings son objetos constantes y por lo tanto muy baratos para el sistema. La mayoría de las funciones relacionadas con cadenas esperan valores String como argumentos y devuelven valores String.

Hay que tener en cuenta que las funciones estáticas no consumen memoria del objeto, con lo cual es más conveniente usar Character que char. No obstante, char se usa, por ejemplo, para leer ficheros que están escritos desde otro lenguaje.

Existen muchos constructores para crear nuevas cadenas:

String();

String( String str );

String( char val[] );

String( char val[],int offset,int count );

String( byte val[],int hibyte );

String( byte val[],int hibyte,int offset,int count );

Tal como uno puede imaginarse, las cadenas pueden ser muy complejas, existiendo muchas funciones muy útiles para trabajar con ellas y, afortunadamente, la mayoría están codificadas en la clase **String**.

### *Funciones Básicas*

La primera devuelve la longitud de la cadena y la segunda devuelve el carácter que se encuentra en la posición que se indica en indice:

int length();

char charAt( int indice );

#### Funciones de Comparación de Strings

boolean equals( Object obj );

boolean equalsIgnoreCase( Object obj );

Lo mismo que *equals()* pero no tiene en cuenta mayúsculas o minúsculas.

int compareTo( String str2 );

Devuelve un entero menor que cero si la cadena es léxicamente menor que str2. Devuelve cero si las dos cadenas son léxicamente iguales y un entero mayor que cero si la cadena es léxicamente mayor que str2.

#### Funciones de Comparación de Subcadenas

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

boolean regionMatch( boolean ignoreCase,int thisoffset,String s2,

int s2offset,int 1 );

Comprueba si una región de esta cadena es igual a una región de otra cadena.

boolean startsWith( String prefix );

boolean startsWith( String prefix,int offset );

boolean endsWith( String suffix );

Devuelve si esta cadena comienza o termina con un cierto prefijo o sufijo comenzando en un determinado desplazamiento.

int indexOf( int ch );

int indexOf( int ch,int fromindex );

int lastIndexOf( int ch );

int lastIndexOf( int ch,int fromindex );

int indexOf( String str );

int indexOf( String str,int fromindex );

int lastIndexOf( String str );

int lastIndexOf( String str,int fromindex );

Devuelve el primer/último índice de un carácter/cadena empezando la búsqueda a partir de un determinado desplazamiento.

String substring( int beginindex );

String substring( int beginindex,int endindex );

String concat( String str );

String replace( char oldchar,char newchar );

String toLowerCase();

String toUpperCase();

String trim();

Ajusta los espacios en blanco al comienzo y al final de la cadena.

void getChars( int srcBegin,int srcEnd,char dst[],int dstBegin );

void getBytes( int srcBegin,int srcEnd,byte dst[],int dstBegin );

String toString();

char toCharArray();

int hashCode();

### *Funciones ValueOf*

La clase **String** posee numerosas funciones para transformar valores de otros tipos de datos a su representación como cadena. Todas estas funciones tienen el nombre de *valueOf*, estando el método sobrecargado para todos los tipos de datos básicos.

Veamos un ejemplo de su utilización:

String Uno = new String( "Hola Mundo" );

float f = 3.141592;

String PI = Uno.valueOf( f );

String PI = String.valueOf( f ); // Mucho más correcto

#### Funciones de Conversión

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 valueOf( char data[] );

String valueOf( char data[],int offset,int count );

Usa arrays de caracteres para la cadena.

String copyValueOf( char data[] );

String copyValueOf( char data[],int offset,int count );

Crea un nuevo array equivalente para la cadena.

## *LA CLASE StringBuffer*

Java posee gran capacidad para el manejo de cadenas dentro de sus clases **String** y **StringBuffer**. Un objeto *String* representa una cadena alfanumérica de un valor constante que no puede ser cambiada después de haber sido creada. Un objeto *StringBuffer* representa una cadena cuyo tamaño puede variar.

La clase **StringBuffer** dispone de muchos métodos para modificar el contenido de los objetos StringBuffer. Si el contenido de una cadena va a ser modificado en un programa, habrá que sacrificar el uso de objetos String en beneficio de StringBuffer, que aunque consumen más recursos del sistema, permiten ese tipo de manipulaciones.

Al estar la mayoría de las características de los StringBuffers basadas en su tamaño variable, se necesita un nuevo método de creación:

StringBuffer();

StringBuffer( int len );

StringBuffer( String str );

Se puede crear un StringBuffer vacío de cualquier longitud y también se puede utilizar un String como punto de partida para un StringBuffer.

StringBuffer Dos = new StringBuffer( 20 );

StringBuffer Uno = new StringBuffer( "Hola Mundo" );

### *Cambio de Tamaño*

El cambio de tamaño de un StringBuffer necesita varias funciones específicas para manipular el tamaño de las cadenas:

int length();

char charAt( int index );

void getChars( int srcBegin,int srcEnd,char dst[],int dstBegin );

String toString();

void setLength( int newlength );

void setCharAt( int index,char ch );

int capacity();

void ensureCapacity( int minimum );

void copyWhenShared();

Obervar que una de las funciones devuelve una cadena constante normal de tipo String. Este objeto se puede usar con cualquier función String, como por ejemplo, en las funciones de comparación.

### *Modificación del Contenido*

Para cambiar el contenido de un StringBuffer, se pueden utilizar dos métodos: *append()* e *insert()*.

En el ejemplo [CadAppend.java](http://proton.ucting.udg.mx/tutorial/java/Fuentes/Cap3/CadAppend.java), vemos el uso de estos dos métodos:

class CadAppend {

public static void main( String args[] ) {

StringBuffer str = new StringBuffer( "Hola" );

str.append( " Mundo" );

System.out.println( str );

}

}

En este otro ejemplo, [CadInversa.java](http://proton.ucting.udg.mx/tutorial/java/Fuentes/Cap3/CadInversa.java), mostramos un método muy simple que le da la vuelta a una cadena:

class CadInversa {

public static String cadenaInversa( String fuente ) {

int longitud = fuente.length();

StringBuffer destino = new StringBuffer( longitud );

for( int i=(longitud-1); i >= 0; i-- )

destino.append( fuente.charAt( i ) );

return( destino.toString() );

}

public static void main( String args[] ) {

System.out.println( cadenaInversa( "Hola Mundo" ) );

}

}

Las funciones que cambian el tamaño son pues:

StringBuffer append( Object obj );

StringBuffer append( String str );

StringBuffer append( char str[] );

StringBuffer append( char str[],int offset,int len );

StringBuffer append( boolean b );

StringBuffer append( int i );

StringBuffer append( long l );

StringBuffer append( float f );

StringBuffer append( double d );

StringBuffer append( char ch );

StringBuffer insert( int offset,Object obj );

StringBuffer insert( int offset,String str );

StringBuffer insert( int offset,char str[] );

StringBuffer insert( int offset,boolean b );

StringBuffer insert( int offset,int i );

StringBuffer insert( int offset,long l );

StringBuffer insert( int offset,float f );

StringBuffer insert( int offset,double d );

StringBuffer insert( int offset,char ch );

### *Operadores de Concatenación*

Hay que recordar que los operadores "**+**" y "**+=**" también se pueden aplicar a cadenas. Ambos realizan una concatenación y están implementados con objetos StringBuffer.

Por ejemplo, la sentencia:

String s = "¿Qué" + " tal ?";

es interpretada por el compilador como:

String s =

new StringBuffer().append( "¿Qué" ).append( " tal ?" ).toString();

y se marcaría el StringBuffer para borrarlo ya que el contenido pasa al objeto String. También, la sentencia:

s += " por ahí!";

sería interpretada por el sistema como:

String s =

new StringBuffer().append( s ).append( " por ahí!" ).toString();

y volvería a marcar para borrar el nuevo StringBuffer.