CLASES EN JAVA

Math

La clase Math representa la librería matemática de Java. Las funciones que contiene son las de todos los lenguajes. 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.rint( double )`	
`Math.pow( a,b )`	
`Math.round( x )`	para double y float
`Math.random()`	devuelve un double
`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

EJEMPLO:

```
import java.io.*;
class ClassMath{
  public static void main (String[] argumentos) throws IOException{
  BufferedReader lector=new BufferedReader ( new
InputStreamReader(System.in));
  float dato;
  System.out.print ("Introduce un numero: ");
  dato = Float.parseFloat ( lector.readLine());
  System.out.println ("El valor absoluto de "+ dato + " es: "
+Math.abs(dato));
  System.out.println ("El seno, coseno y tangente en radianes del número
introducido es ");
  System.out.println ("      1. Seno: " + Math.sin(dato));
  System.out.println ("      2. Coseno: " + Math.cos(dato));
  System.out.println ("      1. Tangente: " + Math.tan(dato));
  System.out.println ("El logaritmo natural es: " + Math.log(dato));
  System.out.println ("La raiz cuadrada: " + Math.sqrt(dato));
  }

}
```

```
RESULTADO 1:
Introduce un numero: -45
El valor absoluto de -45.0 es: 45.0

El seno, coseno y tangente en radianes del número introducido es

      1. Seno: -0.8509035245341184

      2. Coseno: 0.5253219888177297

      1. Tangente: -1.6197751905438615

El logaritmo natural es: NaN

La raiz cuadrada: NaN
```

RESULTADO 2:

```
Introduce un numero: 34
El valor absoluto de 34.0 es: 34.0

El seno, coseno y tangente en radianes del número introducido es

      1. Seno: 0.5290826861200238

      2. Coseno: -0.8485702747846052

      1. Tangente: -0.6234989627162255

El logaritmo natural es: 3.5263605246161616

La raiz cuadrada: 5.830951894845301
```

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.

Declaraciones

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

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

Comprobaciones booleanas

```
Character.isLowerCase( char c )
Character.isUpperCase( char c )
Character.isDigit( char c )
Character.isSpace( char c )
Carácter.isLetter ( char 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 crealo antes.

Traslaciones de caracteres

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

Métodos de la clase Character

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

EJEMPLO

```
import java.io.*;
class Ejemplo2{
  public static void main (String[] argumentos) throws IOException{
  DataInputStream  lector=new DataInputStream (System.in);
  char datoCaracter;
  Character objetoCaracter;
  System.out.print("Introduce un caracter: ");
  datoCaracter=(char)lector.readByte();
  if (Character.isLowerCase (datoCaracter)){
        System.out.println ("La letra es minúsculas");
  }else if (Character.isUpperCase (datoCaracter)){
         System.out.println ("La letra es mayusculas");
  }
  objetoCaracter = new Character (datoCaracter);
  System.out.println ("El valor del objeto Caracter es: " + objetoCaracter);

  }

}
```

```
RESULTADO 1
Introduce un caracter: R
La letra es mayusculas
El valor del objeto Caracter es: R

RESULTADO 2:
Introduce un caracter: d
La letra es minúsculas
El valor del objeto Caracter es: d
```

Float

Cada tipo numérico tiene su propia clase de objetos. Así el tipo *float* tiene el objeto *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*.

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

```
boolean b = F.equals( Object obj );
```

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();
```

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 I.parseInt( String s,int base );
int I.parseInt( String s );
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 );
```

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

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();
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.**

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 );
```

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, los mas utilizados quizas son:

```
String();
String( String str );
```

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.

```
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.

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 );
```

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.