

# **J2SE**

## API de Java

Antonio Espín Herranz

# APIs JAVA

- Versión 7, 8, 9, 10:
  - <http://docs.oracle.com/javase/7/docs/api/>
  - <http://docs.oracle.com/javase/8/docs/api/>
  - <http://docs.oracle.com/javase/9/docs/api/>
  - <http://docs.oracle.com/javase/10/docs/api/>

# La API de Java

Java™ Platform  
Standard Ed. 7  
DRAFT ea-b66

[All Classes](#)

Packages

[java.applet](#)

[java.awt](#)

[ObjectInputValidation](#)

[ObjectOutput](#)

[ObjectStreamConstants](#)

[Serializable](#)

Classes

[BufferedInputStream](#)

[BufferedOutputStream](#)

[BufferedReader](#)

[BufferedWriter](#)

[ByteArrayInputStream](#)

[ByteArrayOutputStream](#)

[CharArrayReader](#)

[CharArrayWriter](#)

[Console](#)

[DataInputStream](#)

[DataOutputStream](#)

[File](#)

[FileDescriptor](#)

[FileInputStream](#)

[FileOutputStream](#)

[FilePermission](#)

[FileReader](#)

[FileWriter](#)

Please note that this documentation is not final and is subject to change.

[Overview](#) [Package](#) **[Class](#)** [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

Java™ Platform  
Standard Ed. 7  
DRAFT ea-b66

java.io

## Class FileInputStream

[java.lang.Object](#)

└ [java.io.InputStream](#)

└ [java.io.FileInputStream](#)

All Implemented Interfaces:

[Closeable](#)

### DESCRIPCION DE LA CLASE:

**Jerarquia de Herencia, que representa la clase, listado de campos, constructores y un listado de todos los campos.**

**Métodos que puede acceder por medio de la herencia.**

```
public class FileInputStream
extends InputStream
```

A `FileInputStream` obtains input bytes from a file in a file system. What files are available depends on the host environment.

`FileInputStream` is meant for reading streams of raw bytes such as image data. For reading streams of characters, consider using `FileReader`.

Since:

JDK 1.0

# La API de Java

- Para trabajar con java es fundamental saber leer correctamente la API. Sobre todo si tenemos que trabajar con alguna clase que no estemos muy familiarizados.
- Trabajar con entornos como Eclipse o Netbeans nos facilitará este trabajo.

# La API de Java

---

**String**( )

Initializes a newly created `String` object so that it represents an empty character sequence.

---

**String**(byte[] bytes)

Constructs a new `String` by decoding the specified array of bytes using the platform's default charset.

---

**String**(byte[] bytes, [Charset](#) charset)

Constructs a new `String` by decoding the specified array of bytes using the specified [charset](#).

---

**String**(byte[] ascii, int hibyte)

**Deprecated.** *This method does not properly convert bytes into characters. As of JDK 1.1, the preferred use the platform's default charset.*

---

**String**(byte[] bytes, int offset, int length)

Constructs a new `String` by decoding the specified subarray of bytes using the platform's default charset.

---

- Cuando un método está catalogado como ***Deprecated*** implica que este método tiende a desaparecer en una nueva versión de java.
- Es mejor no utilizarlos, nos pueden dar problemas en el futuro.

# La API de Java

boolean	<a href="#"><code>contentEquals</code></a> ( <a href="#"><code>StringBuffer</code></a> sb) Compares this string to the specified <code>StringBuffer</code> .
static <a href="#"><code>String</code></a>	<a href="#"><code>copyValueOf</code></a> (char[] data) Returns a <code>String</code> that represents the character sequence in the array specified.
static <a href="#"><code>String</code></a>	<a href="#"><code>copyValueOf</code></a> (char[] data, int offset, int count) Returns a <code>String</code> that represents the character sequence in the array specified.
boolean	<a href="#"><code>endsWith</code></a> ( <a href="#"><code>String</code></a> suffix) Tests if this string ends with the specified suffix.
boolean	<a href="#"><code>equals</code></a> ( <a href="#"><code>Object</code></a> anObject) Compares this string to the specified object.

- La API nos muestra por cada método:
  - Los parámetros que necesita y de que tipo.
  - El tipo que devuelve.
  - Y las posibles excepciones que lanza un método.
  - También nos indica con `static` si el método es estático o no. (*En caso de serlo para utilizarlo sería con el nombre de la clase, no haría falta un objeto de dicha clase.*)

# La API de Java

- Ejemplo: Método static:
- En la API vemos:

**static String copyValueOf(char []data)**

- Este método me dice que devuelve un objeto String y recibe un array de caracteres, y además es estático.

String s; // El objeto que almacena el resultado del método.

char texto[] = {'h','o','l','a'}; // El array de caracteres.

s = **String**.copyValueOf(texto); // Como es static utilizo el  
// nombre de la clase: String.

System.out.println("s: " + s);

# La API de Java

- ¿Qué ocurre cuando el método no es static?
  - Tenemos que crear un objeto para utilizarlo.

## **boolean endsWith(string str)**

- Este método me dice que devuelve un boolean y recibe un String pero ahora no es static.

- Ejemplo:  
String s2;

```
s2 = new String("foto.jpg"); // Creamos el objeto String
// OJO, sólo en la clase String se permite esto también: String s2 = "foto.jpg",
// es una excepción por el gran uso que tiene este objeto.
```

```
if (s2.endsWith("jpg")) // objeto.metodo.
System.out.println("Se trata de un archivo de imagen");
```



# La API de Java

- Un error muy típico que se comete al principio es el siguiente:
  - Cuando ven esto en la API:  
**`boolean endsWith(string str)`**
  - Creen que es obligatorio poner string delante de str y escriben:  
`s2.endsWith(String str)`  
**// NO se puede tomar la API literalmente, me  
// dice que recibe un objeto de TIPO String.**

# La API de Java

- Las Clases de Java están organizadas en paquetes:
  - java.lang: Object, Thread, Exception, Envolveres.
    - El paquete java.lang se incluye por defecto. El resto de paquetes los tendremos que importar cuando queramos utilizar una clase de ellos.
  - java.applet: para los Applets.
  - java.awt: Interface gráfico.
  - java.io: Entrada / Salida.
  - java.net: Trabajo sobre red y protocolo TCP/IP.
  - java.util: Estructuras, Hash.
  - java.swing: Interface gráfico mejorado.

# Paquetes

- Nuestras clases también se pueden organizar en paquetes.
- Un paquete es una ubicación física donde ubicar los .class.
- Podemos crear nuestros propios paquetes o incluir los paquetes de la API de Java.
  - Si creamos un paquete en otra ubicación tenemos que modificar los classpath.
  - `set classpath=%classpath%;c:\java\mis_clases`
- En Java el paquete **java.lang** se incorpora por defecto.

# Paquetes

- Para incluir un paquete utilizamos la sentencia `import nombre_paquete.nombre_clase;`
- Normalmente `import nombre_paquete.*;`
  - Incluimos todas las clases del paquete.
- Podríamos usar una clase sin incluir su paquete pero es mas engorroso, ejemplo:
  - `java.util.StringTokenizer str = new java.util.StringTokenizer();`
  - O incluimos el paquete: `java.util.*;`
  - `StringTokenizer str = new StringTokenizer();`

# Sintaxis

- Ejemplo:

```
package paquete1;  
public class MiClase {  
    ...  
}
```

// También podemos incluir otros paquetes.

```
package paquete1;  
import paquete2.otraClase;  
public class MiClase {  
    ...  
}
```

# Estructura de los paquetes

- Podemos crear la estructura que queramos para generar los paquetes
- Por cada . indicamos un nivel de directorio. package paquete1.paquete2.\*;
  - paquete1 <DIR>
    - paquete2 <DIR>
      - Clase1.class
      - Clase2.class
- Para compilar: javac -d . \*.java

***Prácticas: Paquetes***

# **LAS CADENAS EN JAVA**

## **String, StringBuffer, StringTokenizer**

# String

- Se encuentra en el paquete java.lang
- Constructores:
  - String(), String(String original), String(char []c)
  - Como caso especial podemos usar los String como literales de cadena.

## **Ejemplo:**

String cadena = new String("Hola"); // Válido.

String cadena = "Hola"; // Válido.

// java crea automáticamente un objeto String.



# String

- Podemos usar el objeto String o un literal en los mismo sitios:

```
str.length()           // Devuelve la longitud.  
"hola".length()       // Ok.
```

- Concatenación de cadenas, con el operador +

```
String nombre = "Ana";  
String s = "hola " + nombre + ", ¿Qué tal?";
```

- Podemos concatenar texto con enteros:

```
int edad = 15;  
System.out.println(nombre + " tiene " + edad + " años");  
System.out.println(nombre + " tiene " + edad + 1 + " años");  
System.out.println(nombre + " tiene " + (edad + 1) + " años");
```

# String

- El método `toString()`, devuelve un `String`. Esta clase se encuentra definida en `Object` y todas las clases lo redefinen, es una representación en texto del objeto.
- Las clases que implementemos se suele redefinir este método, basta con devolver un `String` formado por las cadenas que nos interesen.

// Un ejemplo dentro de la clase `Persona`:

```
public String toString(){  
    return(nombre + ", " + apellidos + " → " + edad);  
}
```

```
Persona p = new Persona("Juan", "Martín", 45);  
System.out.println(p.toString());
```

# Métodos de String

## De caracteres:

```
char charAt(int);  
char[] toCharArray();
```

## Comparación:

```
boolean equals(Object);  
boolean equalsIgnoreCase(String);  
== // ¿Es lo mismo que equals?  
int compareTo(String); // 0, < 0, > 0.
```

## Búsqueda:

```
int indexOf(String);  
int lastIndexOf(String);
```

## Modificación de una cadena:

```
String substring(int);  
String concat(String) → +
```

## Conversión de datos:

```
static String valueOf(double) ... (long)
```

***PRÁCTICA: String***

# StringBuffer

- String: representa cadena de longitud fija.
- StringBuffer: representa cadenas que pueden crecer y sobre escribirse. Y se pueden insertar caracteres o subcadenas en el lugar que nos interese.
- Tiene 3 constructores:
  - StringBuffer() → Reserva para 16 car.
  - StringBuffer(int tamaño) → Reserva según tamaño.
  - StringBuffer(String str) → Inicializa con str y además reserva para otros 16 car.

# Métodos de StringBuffer

- `length()` y `capacity()`: la longitud y la capacidad total.
- `ensureCapacity(int capacidad)` → preestablece la capacidad del buffer.
- `setLength(int nuevaLong)` → Establece una nueva longitud del buffer, si acortamos se pierde.
- `charAt(int donde)` / `setCharAt(int donde, char car)` → Extraer y establecer car.
- `getChars(int ini, int fin, char destino[], int iniDest)` → Extrae caracteres a un array, controlar que destino sea suficientemente grande.

# Métodos de StringBuffer

- `StringBuffer append(String / int / Object)` → Devuelve el mismo buffer, podemos anidar llamadas.
- `StringBuffer insert(int, String / char / Object)` → Para insertar en el buffer en una posición.
- `StringBuffer delete (int ini, int fin) / StringBuffer deleteCharAt(int)`
- `StringBuffer replace(int posIni, int posFin, String str)`
- `String substring(int posIni)`

# StringTokenizer

- Paquete `java.util`
- Proporciona uno de los primeros pasos para realizar un análisis gramatical de una cadena de entrada, extrayendo los símbolos que se encuentren en esa cadena.
- String de entrada y un String de delimitación. Los delimitadores marcan la separación entre los símbolos que se encuentran en la cadena de entrada.

# Métodos de StringTokenizer

- `StringTokenizer(String str, String delim)`
- `int countTokens()` → Devuelve el número de Tokens.
- `boolean hasMoreElements()` → ¿Quedan tokens?
- `boolean hasMoreTokens ()` → ¿Quedan tokens?
- `String nextToken()` → Devuelve el próximo Token.
- `String nextToken(String delim)` → Devuelve el próximo token a partir del nuevo delimitador.

***Práctica StringTokenizer***



# **LAS CLASES ENVOLVENTES**

# Las Clases Envolvertes

- Permiten representar tipos básicos de datos, como objetos.

<u>Tipo de dato básico</u>	<u>Clase envolverte</u>
boolean	Boolean
byte	Byte
char	Character
short	Short
int	Integer
long	Long
float	Float
double	Double

# Character

- Representa la clase envolvente del tipo primitivo: **char**
- Constructor: **Character(char valor)**
- Comprobaciones booleanas (static):
  - Character.isLowerCase( c )
  - Character.isUpperCase( c )
  - Character.isDigit( c )
  - Character.isSpace( c )

# Métodos de Character

- Conversiones de mayúsculas / minúsculas:
  - `char c2 = Character.toLowerCase( c );`
  - `char c2 = Character.toUpperCase( c );`
- Conversiones entre carácter / dígito:
  - `int i = Character.digit( c,base );`                      `// static`
  - `char c = Character.forDigit( i,base );`                      `// static`
- Más métodos:
  - `C = new Character( 'J' );`
  - `char c = C.charValue();`
  - `String s = Character.toString();` `// static`

# Float

- Envuelve al tipo primitivo **float**.
- Constructores:
  - Float(float valor);
  - Float(double valor);
  - Float(String valor);
- Valores de Float:
  - Float.POSITIVE\_INFINITY → Representa el valor + infinito.
  - Float.NEGATIVE\_INFINITY → Representa el valor – infinito.
  - Float.NaN → Indica que es un No Número.
  - Float.MAX\_VALUE → Máximo valor del Float.
  - Float.MIN\_VALUE → Mínimo valor del Float.

# Métodos de Float

- Comprobaciones:
  - `boolean b = Float.isNaN( f );` → Devuelve true si es un No Número.
  - `boolean b = Float.isInfinite( f );` → Devuelve true si es infinito.
- Conversiones con cadenas:
  - `String s = Float.toString( f );`
  - `Float f1 = Float.valueOf( "3.14" );`
  - `float f2 = Float.parseFloat("3.14");`
- Conversiones de objeto: // Devuelve representación de Float en los tipos primitivos:
  - `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();`

# Métodos de Float

- `int i = F.hashCode();`
  - Devuelve el código Hash correspondiente al valor Float.
- `boolean b = F.equals( Object obj );`
  - Compara el Float con un objeto.
- `static int compare(Float f1, Float f2);`
  - Compara dos float,
    - $0 \rightarrow$  iguales,
    - $1 \rightarrow f1 > f2$ ,
    - $-1 \rightarrow f1 < f2$
- `int compareTo(Float otroFloat);`

# Double

- Envuelve al tipo primitivo **double**.
- Constructores:
  - Double(double valor);
  - Double(String valor);
- Valores de Double:
  - Double.POSITIVE\_INFINITY → Representa el valor + infinito.
  - Double.NEGATIVE\_INFINITY → Representa el valor – infinito.
  - Double.NaN → Indica que es un No Número.
  - Double.MAX\_VALUE → Máximo valor del Double.
  - Double.MIN\_VALUE → Mínimo valor del Double.
- Métodos: igual que en la clase Float, pero para Double.



# Integer

- Envuelve al tipo primitivo: **int**.
- Valores:
  - MIN\_VALUE: El mínimo entero.
  - MAX\_VALUE: El máximo entero.
- Constructores:
  - Integer(int valor);
  - Integer(String valor);

# Métodos de Integer

- `int compareTo(Integer otroInteger);`
- `static int parseInt(String valor);`
- `int intValue();`
- `String toString();`
- `toBinaryString();`
- `toOctalString();`
- `toHexString();`
- `static Integer valueOf(String);`

# Long

- Envuelve al tipo primitivo: **long**.
- Mismos constructores y métodos que en la clase Integer pero para Long.
- `static long parseLong(String valor);`
- `String toString();`

# Boolean

- Envuelve al tipo primitivo: **boolean**
- Valores:
  - static Boolean FALSE, TRUE
- Constructores:
  - Boolean(boolean value);
  - Boolean(String valor);
- Métodos:
  - boolean booleanValue();
  - String toString();
  - static Boolean valueOf(String s);

# Otras clases de la API

# Math

- Está en java.lang → No hay que indicarlo, se carga por defecto.
- Todos los métodos son static → NO necesito un objeto de la clase para utilizarlos.
- Sintaxis → `Math.pow(2, 3);` → 8
- Dos constantes: PI y el número E.
- Cálculos de trigonometría: sin, cos, tan.
- Redondeos: round, floor, ceil
- Raíz cuadrada: sqrt
- Potencia: pow
- max y min.

# Random

- Se utiliza para generar números de forma aleatoria.
- Está en `java.util`.
- `Random()` y `Random(long semilla)` → Crean un generador de números aleatorios.
- `nextBoolean()`, `nextInt()`, `nextDouble()`, `nextLong()`
- `setSeed(long semilla)` → Establecer el valor de la semilla.

# Date

- Está en `java.util`
- `Date()` → Fecha y hora actuales.
- `Date(long milisegundos)` → una fecha en milisegundos a partir del 1 de enero de 1970.
- `setTime(long milisg)`
- `toString()` → Devuelve en formato `yyyy\mm\dd`
- `static Date valueOf(String s)` → `s` será de la forma `yyyy-mm-dd`, devuelve esta fecha formateada para `java.sql.Date`



# Ejemplo: Calendar, DateFormat

```
Calendar calendar;
```

```
Date fecha;
```

```
fecha = new Date();
```

```
calendar = Calendar.getInstance();
```

```
calendar.setTime(fecha);
```

```
System.out.println(calendar);
```

```
System.out.println(calendar.get(Calendar.DAY_OF_MONTH) + "/" +  
    (calendar.get(Calendar.MONTH)+1)+"/"+calendar.get(Calendar.YEAR));
```

```
DateFormat dt = DateFormat.getInstance();
```

```
System.out.println(dt.format(fecha));
```

# Convertir Fecha a Calendar

```
public static Calendar getCalendar(String fecha){  
    // Devuelve la fecha recibida como un Calendar. La fecha viene en el formato: dd-mm-yyyy.  
    Calendar c = Calendar.getInstance();  
  
    // Extraemos el año.  
    int y1 = Integer.parseInt(fecha.substring(6));  
  
    // Extraemos el mes.  
    String cadenaM1 = fecha.substring(3, 5);  
    if (cadenaM1.startsWith("0")){  
        cadenaM1 = cadenaM1.substring(1);  
    }  
    int m1 = Integer.parseInt(cadenaM1)-1; // los meses los toma de 0..11  
  
    // Extraemos el día.  
    String cadenaD1 = fecha.substring(0, 2);  
    if (cadenaD1.startsWith("0")){  
        cadenaD1 = cadenaD1.substring(1);  
    }  
    int d1 = Integer.parseInt(cadenaD1);  
    c.set(y1, m1, d1);  
  
    return(c);  
}
```

# SimpleDateFormat

Formatear fechas con SimpleDateFormat:

```
public static String getFecha(){  
    SimpleDateFormat d = new SimpleDateFormat("dd/MM/yyyy");  
    return d.format(new Date());  
}
```