Algunas clases útiles de la librería de Java

Este documento hace un resumen somero de cuatro clases útiles de la librería de Java (`Scanner`, `String`, Envoltorios y `Math`). Sólo se comentan algunos de los métodos más interesantes de estas clases, para un análisis más detallado se recomienda visitar la API de Java.

1. Clase `Scanner`

Esta clase se usa entre otras cosas para que el usuario pueda introducir datos por el teclado. Para usarla hay que importarla desde la librería `java.util`. A continuación se muestra un ejemplo de código con su uso (se omite la declaración de la clase y del método main):

```
import java.util.Scanner;
...
Scanner s = new Scanner(System.in);
int b=0;
while (!s.hasNextInt())
      s.next();
b= s.nextInt();
System.out.println(b);
```

Por defecto los datos se separan mediante un espacio. Se puede cambiar el carácter separador, por ejemplo para cambiarlo por Enter usamos el método: `<objeto scanner>.useDelimiter(System.getProperty("line.separator"));`

Se pueden usar otros delimitadores e incluso combinaciones de ellos por medio de expresiones regulares. En internet se pueden encontrar más ejemplos.

También por defecto, si el teclado del ordenador está en español, se usa la coma en lugar del punto para los números decimales. Si queremos que el usuario introduzca los datos usando el punto utilizamos (hay que importar la clase `Locale` de `java.util`): `<objeto scanner>.useLocale(Locale.ENGLISH);`

Algunos de los métodos más interesantes de esta clase son:

Nombre método	Descripción
`int nextInt()`	Devuelve un valor de tipo `int` que el usuario debe introducir por teclado. Hay un método para cada tipo básico, excepto para `char`. Ej. `nextDouble()`, `nextBoolean()`...
`String next()`	Devuelve el siguiente dato introducido por el usuario en forma de `String`
`String nextLine()`	Devuelve todo lo que ha introducido el usuario, independientemente de cuál sea el separador (da problemas al usarse, mejor no utilizarlo)
`boolean hasNext()`	Devuelve `true` si hay algún dato listo para ser leído
`boolean hasNextInt()`	Devuelve `true` si lo siguiente que va a leer es un `int` (si lo siguiente que ha introducido el usuario es un `int`). Hay un método similar para cada tipo básico, excepto `char`. Ej. `hasNextDouble()`, `hasNextBoolean()`...

2. Clase `String`

String, además de comportarse como un tipo básico es una clase, por lo que tiene métodos que se pueden utilizar para hacer operaciones con cadenas. Para llamar a los métodos se pone <variable de tipo String>.metodo

Nombre método	Descripción
`char charAt(int index)`	Devuelve el carácter que está en esa posición
`int compareTo(String anotherString)`	Devuelve 0 si ambas cadenas son iguales, un valor negativo si la cadena es anterior alfabéticamente que el argumento y un valor positivo si es mayor. Si son diferentes devuelve la diferencia de código ASCII entre las dos primeras letras en que se diferencian. Si solo se diferencian en que son de distinta longitud lo que devuelve es la diferencia en la longitud. ¡Ojo la ñ no va después de la n!
`int compareToIgnoreCase(String str)`	Igual al anterior ignorando la diferencia entre mayúsculas y minúsculas
`boolean contains (charSequence s)`	Devuelve `true` si la cadena contiene a la subcadena
`boolean endsWith(String suffix)`	Devuelve `true` si la cadena acaba de esa forma. Hay otro equivalente si la cadena empieza de esa forma (`startsWith`)
`boolean equals (String str)` `boolean equalsIgnoreCase(String str)`	Devuelve verdadero si las dos cadenas son iguales, en el segundo caso ignorando mayúsculas y minúsculas.
`int indexOf(String str)` `int indexOf(String str, int ind)`	Devuelve un entero con la posición en la que aparece el carácter o la subcadena por primera vez (-1 si no existe). La segunda versión empieza a buscar desde un lugar determinado. También hay 2 versiones equivalentes buscando de atrás hacia delante (`lastIndexOf`)
`int length()`	Devuelve la longitud de la cadena
`String replace (String , String)`	Reemplaza todas las ocurrencias de una subcadena por otra. También se puede usar `replaceAll` o `replaceFirst`. La primera se comporta igual pero además de recibir una cadena puede recibir una expresión regular. La segunda, también puede recibir expresiones regulares, y solo cambia la primera ocurrencia de la subcadena.
`String[ ] split (String regex)`	Devuelve un array de `String` resultado de partir la cadena usando como separador el argumento (cadena o expresión regular). Ej. `"hola como estás".split(" ")` ␣ `new String[] {"hola","como","estás"}`

`String substring(int beginIndex, int endIndex)` `String substring(int beginIndex)`	Devuelve la subcadena que empieza en el índice pasado por parámetro (incluido), si solo se le da un parámetro devuelve desde el índice hasta el final de la cadena original, si tiene dos devuelve entre los dos índices, con el primero incluido y el segundo excluido.
`String toLowerCase()`	Convierte la cadena a minúsculas, y `toUpperCase` a mayúsculas.
`static String valueOf(tipo básico o char [])`	Convierte el tipo básico que se le pase a `String`
`String trim ()`	Elimina espacios antes y después

Los métodos no cambian el valor de la cadena (si hacemos por ejemplo un `cadena.toLowerCase()`, la cadena original no cambia).

En `replaceAll`, `replaceFirst` y `split` los siguientes caracteres no se pueden reemplazar directamente: `$ ^ . * + ? [ ]()`. Hay que poner `\\?` y similares (la razón es que no se busca una cadena sino una expresión regular)

3. Envoltorios

Los envoltorios se utilizan cuando queremos guardar datos de tipos básicos como si fueran objetos (hay situaciones en las que Java precisa que el dato esté en forma de objeto y no admite tipos básicos). Para cada tipo básico existe un envoltorio:

Tipo básico	**Envoltorio**
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`

En cualquier caso Java es capaz de empaquetar/desempaquetar automáticamente los datos, de forma que si en algún sitio se necesita un tipo objeto y en lugar de ello se proporciona un tipo básico, se convierte automáticamente el básico a su correspondiente envoltorio y viceversa.

Además, los envoltorios permiten convertir de String al tipo básico correspondiente, para ello tienen un método `parse`, (`parseInt()`, `parseBoolean()`, `parseFloat()` …) que funciona como en el siguiente código:

```
String s = "33";
int a = Integer.parseInt(s); //a vale 33
```

Si el valor que hay en la cadena no se puede convertir al tipo de destino, tenemos un error de ejecución

4. Clase `Math`

Clase especial que contiene funciones y constantes matemáticas.

Se usa poniendo `Math.<metodo>`

Atributos: `Math.E` y `Math.PI`

Nombre método	Descripción
`static int abs(int a)`	Devuelve el valor absoluto del número pasado como parámetro. También se puede usar con cualquier otro tipo numérico (si le damos un `double` devolverá un `double`, etc.)
`static long round (double a)`	Redondea el número pasado como parámetro. Si el número es `double` devuelve `long`, si es `float` devuelve `int`.
`static double ceil (double a)`	Trunca el número hacia arriba (`Math.ceil(3.2)` devuelve `4.0`), ¡ojo devuelve un `double`!
`static double floor (double a)`	Trunca el número, ¡ojo devuelve un `double`!
`static double sin (double a)`	Devuelve el seno del ángulo `a` (`a` debe estar en radianes). También hay `cos`, `tan`, `asin`, `acos`, `atan`, `sinh`, `cosh`, `tanh`
`static int max (int a, int b)`	Devuelve el máximo de los dos números. También hay versiones para los otros tipos numéricos. También existe `min (int a, int b)`.
`static double log (double a)`	Devuelve el logaritmo neperiano de `a`, para el logaritmo decimal se usa `log10(double a)`
`static double pow (double a, double b)`	Eleva `a` a `b`
`static double exp (double a)`	Eleva el número `e` a `a`
`static double sqrt (double a)`	Raíz cuadrada
`static double cbrt (double a)`	Raíz cúbica
`static double random()`	Devuelve un número aleatorio entre `0.0` (incluido) y `1.0` (no incluido)