Algunas clases de utilidad

String

- Recordar que para comparar String no hay que utilizar == sino `equals(Object o)`
- Si queremos que no se distinga entre mayúsculas y minúsculas `equalsIgnoreCase()`
- La longitud de la cadena se obtiene con `length()`
- `public String toLowerCase()` nos dará la cadena en minúsculas (`toUpperCase()` mayúsculas)
- Posee el método sobrecargado static `String valueOf(boolean/char/int/long/float/double)` para convertir tipos primitivos a cadenas.
- Podemos formatear cadenas con format, (como hacíamos con printf ==format)

  Ej: `String formateada = String.format("El valor de PI es: %2.2f ", 3.1415);`
- Método interesante: `String split(String s)` que divide la cadena buscando un patrón.

  Ej:
```
String entrada = "Esta cadena, tiene comas, y ahí se irá partiendo.";
String[] partes = entrada.split(", ");
for (String parte:partes) System.out.println(parte)
```

StringBuffer y StringBuilder

- La clase String es poco eficiente cuando se trata de manipular cadenas de caracteres. Las crea inmutables y si hay que modificarlas debe crear nuevas cadenas.
- Ej: `String concatenaCadenadas = “Hola” + “, que tal”;`
- Se han creado tres objetos de tipo String.
- Por tanto cuando se vayan a hacer manipulaciones continuadas sobre las cadenas es preferible utilizar una clase más especializada.

StringBuffer es igual que String , pero no es inmutable. Más eficiente en el caso de ir a hacer concatenaciones…

Para concatenar utiliza el método sobrecargado append(boolean/int/…/String/StringBuffer).

Otros métodos: int indexOf(String s) devuelve la posición de la primera ocurrencia de s

insert(int offset, boolean/char/.../String) inserta en offset el argumento.

StringBuffer está sincronizado (uso de hilos). Si no necesitamos que esté sincronizado, se comporta igual StringBuilder pero más optimizada.

Clases recubridoras

Permiten utilizar tipos primitivos como objetos. Para cada tipo primitivo hay una clase recubridora.

Se puede hacer uso del Autoboxing, en el que se convierte de modo automático y transparente tipos primitivos a clases recubridoras siempre que sean compatibles con el correspondiente unboxing.

Ej:
```
Integer entero = 15;

int enteroPrimitivo = entero;
```

Para transformar cadenas hay que parsear.

```
int entero = Integer.parseInt(“20”);
```

Trabajo con fechas. La clase Date

Estas clases pertenecen al paquete java.utill

Representa un instante de tiempo con una precisión de milisegundos.

Para obtener el instante de tiempo actual, simplemente creamos una instancia de esta clase a partir de su constructor por defecto. La información es un long con los milisegundos transcurridos desde el 1/1/1970 a las 00:00

Ej: System.out.println(“Ahora: “+new Date());

Muchos de los métodos de esta clase están obsoletos. Para trabajar con fechas, se utiliza la clase GregorianCalendar.

SimpleDateFormat (formateo de fechas, por curiosidad)

Nos permite definir el formato con el que mostrar fechas.

Para ver todas las opciones revisar la API

Los más útiles EEEE muestra el día de la semana, dd el ordinal del día dentro del mes, MMMM el nombre del mes, yyyy el año, hh la hora en formato 24 horas, mm los minutos y ss los segundos.

Ej:

```
        SimpleDateFormat sdf= new SimpleDateFormat("EEEEEE dd 'de' MMMM 'de' yyyy '('hh':'mm':'ss')'");

        System.out.printf("Ahora: "+sdf.format(new Date()));
```

Calendar (obsoleta, por culturilla)

La clase Calendar permite convertir instantes temporales (representados por la clase Date), a una fecha expresada en días, meses, año, horas, minutos y segundos. La clase Calendar es abstracta, por lo que no se puede instanciar. Nos proporciona la funcionalidad mínima para trabajar con fechas, y otras extensiones de esta clase implementan calendarios concretos, como la clase GregorianCalendar, que nos permite, por ejemplo saber qué día de la semana será dentro de 40 días.

```
GregorianCalendar calendario = new GregorianCalendar();

System.out.println("Ahora: "+calendario.getTime());

calendario.add(Calendar.DAY_OF_YEAR, 40);

System.out.println("Dentro de 40 días: "+calendario.getTime());
```

(todas las constantes para indicar el día, mes, etc están definidas en la clase Calendar)

Expresiones regulares en Java

Una expresión regular define un patrón de búsqueda para cadenas de caracteres. La podemos utilizar para comprobar si una cadena contiene o coincide con el patrón. El contenido de la cadena de caracteres puede coincidir con el patrón 0, 1 o más veces.

El patrón se busca en el String de izquierda a derecha. Cuando se determina que un carácter cumple con el patrón, este carácter ya no vuelve a intervenir en la comparación.

Por ejemplo, la expresión regular 010 se encuentra en el String 010101010 dos veces. (010101010)

Símbolos comunes en expresiones regulares

.	Un punto indica cualquier carácter.
^expresión	El símbolo ^ indica el principio del String. En este caso el String debe contener la expresión al principio.
expresión$	El símbolo $ indica el final del String. En este caso el String debe contener la expresión al final.
[abc]	Los corchetes representan una definición de conjunto. En este ejemplo el String debe contener una de las letras a, b ó c.
[abc][12]	El String debe contener una de las letras a, b ó c seguidas de un 1 ó 2
[^abc]	El símbolo ^ dentro de los corchetes indica negación. En este caso el String debe contener cualquier carácter excepto a, b ó c.
[a-z1-9]	Rango. Indica las letras minúsculas desde la a hasta la z (ambas incluidas) y los dígitos desde el 1 hasta el 9 (ambos incluidos)
A|B	El carácter | es un OR. Indica A ó B
AB	Concatenación. A seguida de B

Meta caracteres

\d	Dígito. Equivale a [0-9]
\D	No dígito. Equivale a [^0-9]
\s	Espacio en blanco. Equivale a [\t\n\x0b\r\f]
\S	No espacio en blanco. Equivale a [^\s]
\w	Una letra mayúscula o minúscula, un dígito o el carácter _ Equivale a [a-zA-Z0-9]
\W	Equivale a [^\w]
\b	Límite de una palabra.

En java hay que usar \\. Por ejemplo para \w hay que poner \\w. Para la barra invertida \\\

Cuantificadores

{X}	Indica que lo que va justo antes de las llaves se repite X veces
{X,Y}	Indica que lo que va justo antes de las llaves se repite un mínimo de X veces y máximo Y veces. También podemos poner {X,} que indica un mínimo de X sin límite máximo.
*	Indica 0 ó más veces. Equivale a {0,}
+	Indica 1 ó más veces. Equivale {1,}
?	Indica 0 ó 1 veces.Equivale a {0,1}

Enlace a ejemplos del uso de expresiones regulares

En esta web tenéis la información que os he puesto y más ejemplos.

http://puntocomnoesunlenguaje.blogspot.com/2013/07/ejemplos-expresiones-regulares-java-split.html