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”);
```