## 1. Introducción

Podemos definir las cadenas de texto String como colecciones de char (caracteres). Veamos algunos de los métodos que nos ofrecen los String.

- Longitud: el método `.length()` devuelve la longitud de la cadena.
- Concatenar: el operador `+` sirve para concatenar cadenas. También podemos emplear el método `.concat(String)`, que devuelve la cadena final.
- Carácter en una posición: el método `.charAt(int)` devuelve el carácter en la posición indicada. Es muy útil para recorrer cadenas carácter a carácter.
- Indicar la posición de un carácter: con el método `.indexOf(char)` obtendremos un entero con la primera aparición del carácter indicado. Si no está, devuelve -1.
  Si se quiere obtener la siguiente aparición de un carácter a partir de una posición determinada, se indica como segundo parámetro: `.indexOf(char, int)`.
- Comparación: el método `.equals(String)` devuelve un *boolean* que será *true* si el contenido de dos cadenas es igual, y *false* en caso contrario.
  El método `.equalsIgnoreCase(String)` se utilizará cuando no queremos distinguir entre mayúsculas y minúsculas.
- Ver si está vacía: el método `.isEmpty()` devuelve un *boolean*.
- Convertir en mayúsculas o minúsculas: con los métodos `.toUpperCase()` y `.toLowerCase()`, que devuelven un String, convertimos en mayúsculas y minúsculas todo el texto, respectivamente.
- Para reemplazar un carácter por otro, podemos utilizar el método `.replace(char)`, que devuelve la cadena resultante. También se puede reemplazar una cadena.
- El método `.startsWith(String)` devuelve un boolean indicando si la cadena empieza o no por lo indicado. También se puede indicar a partir de una posición determinada, como segundo parámetro.
- De manera similar al anterior, tenemos el método `.endsWith(String)`.
- Para obtener una subcadena de otra más grande, el método `.substring(int)` devolverá la cadena a partir de esa posición. También se puede indicar entre dos posiciones, añadiendo el índice final como segundo parámetro.
- El método `.trim()` quita los espacios al final y al principio de las cadenas. Es un método muy útil cuando manejamos formularios y almacenamiento en bases de datos, ya que la cadena “Julia “ (con un espacio al final) no es lo mismo que “Julia”.
- Dividir una cadena en subcadenas: `.split()`. Con esta función devolvemos un *array* de Strings con los cortes creados por la cadena que ponemos en el *split*. Lo veremos más adelante, cuando veamos los *arrays*.

## 2. Entradas y salidas estándares

Cuando programamos cualquier aplicación, es muy usual querer mostrar datos e introducirlos. Los métodos para hacerlo son muy variados, pero uno de los más sencillos y comunes serán las entradas y salidas estándares.

Cuando hablamos de entrada y salida estándar, por defecto nos referimos a la consola. En concreto son tres:

- System.out: Salida estándar.
- System.err: Salida estándar de error.
- System.in: Entrada estándar.

Por defecto, las salidas normales y de error son la misma (la consola), pero se utilizan ambas por si en algún momento del desarrollo una de ellas se quiere redireccionar a otro sitio (un fichero de *log*, por ejemplo).

Para escribir por pantalla utilizamos este método:

```
System.out.println("Salida estándar");
System.err.println("Salida de error");
```

Para introducir datos, utilizamos la clase Scanner inicializada de esta forma:

```
Scanner sc = new Scanner(System.in);
```

Y a partir de este momento, podemos utilizar los métodos `.nextInt()`, `.nextDouble()`, `.nextFloat()`, etcétera, para leer por la entrada estándar los tipos concretos deseados.

```
int numero = sc.nextInt();
double decimal = sc.nextDouble();
```

Para evitar errores de lectura entre Strings y el resto de datos primitivos, **se suele utilizar un Scanner para datos primitivos y otro diferente para cadenas de texto**. Así evitamos problemas con el último carácter de salto de línea que se ha introducido por consola. Para leer un String, podemos utilizar el método `.nextLine()`.

```
Scanner scTxt = new Scanner(System.in);
String cadena = scTxt.nextLine();
```

### 2.1. Formatos de visualización

Si queremos configurar cómo se van a mostrar los datos, podemos aplicar una serie de formatos para elegir el tipo de números, la notación, el número de decimales, etc. que queremos utilizar.

Para ello, utilizaremos el método `System.out.printf()`, indicando el formato que queremos dar en cada uno de los casos.

```
System.out.printf("%n");      //%n salto de línea
int entero = 11;
System.out.printf("%d%n", entero);  //Número entero: 11
System.out.printf("n=%d %n", entero);     //Concatena: n=11
System.out.printf("%05d%n", entero);      //Añade 0: 00011
System.out.printf("%5d%n", entero); //Añade espacios:    11
System.out.printf("%o %n", entero); //Octal: 13
System.out.printf("%X %n", entero); //Hexadecimal: B
```

```
double dec = 1234.56789;
System.out.printf("%f%n", dec);     //Decimal: 1234,56789
System.out.printf("%.3f %n", dec); //3 decimales: 1234,568
System.out.printf("%.7f %n", dec); //7 decimales: 1234,5678900
System.out.printf("%10.2f %n", dec);      //10 caracteres en total
(añade espacios delante)
System.out.printf("%e %n", dec);    //Notación científ: 1,234568e+03
System.out.printf("%.2e %n", dec); //2 decimales científ: 1,23e+03


entero = 98;
System.out.printf("%c %n", entero); //Caracter: b
```

Si quieres más información sobre este método, aquí puedes consultar una tabla con todos los formatos, además de ver que también podemos utilizar `String.format()` si queremos reutilizar esos Strings y no solamente imprimirlos por pantalla.