Métodos de la Clase String.

TIPO DEVUELTO	NOMBRE Y SIGNATURA DEL MÉTODO	DESCRIPCIÓN DE SU FUNCIÓN.

El objeto String que crea, ya que es uno de los constructores disponibles.	`String(String original)`	Constructor de un `String` a partir de otro, o de un literal `String`.
		La clase `String` tiene otra docena de constructores, la mayoría para crear un `String` a partir de un array de bytes, o de char. Alguno de ellos será usado más adelante, pero por ahora, con este nos basta.
		RECUERDA:
		`String nombre = "Juan";`
		equivale en todos los aspectos a:
		`String nombre = new String ("Juan");`
		Y
		`String nombre = "";`
		Equivale a
		`String nombre = null;`

char	`charAt(int index)`	Devuelve el carácter (`char`) que se encuentra en la posición que indica el parámetro entero `index`. Debes tener en cuenta que el primer carácter de cada `String` se encuentra en la posición 0 (cero)
		
		Ejemplo:
		`String nombre= "Juan";`
		`char inicial=nombre.charAt(0);`
		
		Como resultado inicial toma el valor `'J'`

boolean	`equals(Object anObject)`	Compara dos `String`, y devuelve verdadero si sus valores son iguales, es decir, si representan a la misma cadena de caracteres.
		
		Ejemplo:
		`String a= "Juana";`
		`String b = "Juanito";`
		`boolean resultado= a.equals(b);`
		
		La variable resultado toma el valor falso (`false`), que es el valor que devuelve el método `equals()` en este caso, ya que a y b representan a cadenas con valores distintos.
		
		¡OJO!: Si comparas dos objetos `String` usando el operador `==,` puedes tener sorpresas, ya que `==` no compara el contenido de los objetos, si no el valor de las referencias. Si las dos referencias son iguales, quiere decir que realmente apuntan al mismo objeto `String`, y por tanto su valor será igual, ya que es el mismo. Pero si son distintas, quiere decir que apuntan a objetos distintos, que pueden ser iguales o no. Por tanto, habrá que usar el método `equals()` o el método `compareTo(),` para comparar los valores de los objetos, en vez de las referencias.
		Fíjate que como no es un método static, debemos ejecutarlo para un objeto, es un mensaje que le enviamos al objeto para que haga algo con él. El objeto al que se le envía, en nuestro ejemplo el `String` a es un parámetro implícito de ese método. De hecho es uno de los dos objetos a comparar. Por último llamar la atención en el hecho de que lo que pide como argumento es un `Object`, que es la clase de la que heredan todas las demás clases en Java. En el fondo, cualquier objeto de cualquier clase que creemos "es un `Object`", por herencia, lo que quiere decir que en cualquier sitio donde esperemos un `Object` en general, podremos colocar cualquier subtipo de Object particular. En concreto, los objetos `String` también son `Object`, por lo que podemos pasarlos como parámetros. En las próximas unidades volveremos sobre el concepto de herencia.

TIPO DEVUELTO	NOMBRE Y SIGNATURA DEL MÉTODO	DESCRIPCIÓN DE SU FUNCIÓN.

boolean	`equalsIgnoreCase(String anotherString)`	Al igual que `equals()`, sirve para comparar dos `String`, pero con la peculiaridad de que no tiene en cuenta si las letras son mayúsculas o minúsculas.
		
		Así:
		`String a = "juan";`
		`String b = "JUAN";`
		`boolean result1=a.equals(b);`
		`boolean result2=a.equalsIgnoreCase(b);`
		`result1` toma el valor false, mientras que `result2` toma el valor true.

int	`compareTo(String anotherString)`	Compara dos objetos `String` lexicográficamente, es decir, básicamente en el orden alfabético que estamos acostumbrados a usar. Si una cadena va alfabéticamente delante, se considera que es menor.
		
		Devuelve un entero negativo si `a` va alfabéticamente delante de `b`.
		Devuelve cero si `a` y `b` son iguales.
		Devuelve un valor positivo si `b` va alfabéticamente delante de `a`.
		
		Ejemplo:
		
		`String nombre = "Manuel Jesús Rubia Mateos";`
		`String nombre2= "Manuel Rubia Mateos";`
		`if(nombre.compareTo(nombre2)<0){`
		`System.out.println (nombre+ " va delante de "+ nombre2);`
		`}else {`
		`System.out.println (nombre2+ " va delante de "+ nombre);`
		`}`
		
		Escribirá el primer mensaje:
		`Manuel Jesús Rubia Mateos va delante de Manuel Rubia Mateos`
		ya que la condición es verdadera. Al comparar la J de Jesús con la R de Rubia, sabemos que la J va delante alfabéticamente, luego se considera que el valor de `nombre` va alfabéticamente delante de `nombre2`
		
		Como resultado del método tendremos un valor negativo, que al compararlo con cero, será menor, y por tanto la condición será verdadera.
		
		¡OJO!: Ten en cuenta que para comparar los valores, tendrás que usar `compareTo()` o `equals()`, pero no el operador `==` que sólo compara las referencias.

int	`compareToIgnoreCase(String str)`	Al igual que `compareTo()`, compara dos `String` alfabéticamente, pero con la diferencia de que no tiene en cuenta si las letras son mayúsculas o minúsculas. Las considera iguales.
		
		`String nombre = "Miguel Ángel Pérez Martínez";`
		`String nombre2= "miguel Ángel pÉrez mARtíNez";`
		`if(nombre.compareToIgnoreCase(nombre2)==0){`
		`System.out.println ("Ambos nombres son iguales.");`
		`}`
		
		El método `compareToIgnoreCase()`, al no distinguir mayúsculas de minúsculas, devuelve cero, ya que considera las dos cadenas iguales.

TIPO DEVUELTO	NOMBRE Y SIGNATURA DEL MÉTODO	DESCRIPCIÓN DE SU FUNCIÓN.

String	`concat(String str)`	Concatena el `String` pasado como argumento (`str`) al objeto `String` al que se le envía el método o mensaje.
		
		Ejemplo:
		`String nombre = "Alfonso ";`
		`String apellidos= "Bonillo Sierra";`
		`nombre = nombre.concat(apellidos);`
		`System.out.println ("Nombre = "+nombre);`
		
		Atención: La misma operación puede hacerse con el operador `+`, que cuando recibe operandos de tipo `String`, también realiza la operación de concatenación.
		
		Un código equivalente sería:
		`String nombre = "Alfonso ";`
		`String apellidos= "Bonillo Sierra";`
		`nombre = nombre + apellidos;`
		`System.out.println("Nombre = ".concat(nombre));`

boolean	`startsWith(String prefix)`	Devuelve `true` (verdadero) si el `String` al que se le envía el mensaje o para el que se ejecuta el método (es lo mismo) comienza con la subcadena indicada en el argumento `prefix`.
		
		Ejemplo:
		`String nombre = "Silvia Thomas Barrós";`
		`if (nombre.startsWith(Silvia)){`
		`  System.out.println("El nombre es Silvia");`
		`}`
		
		Se escribirá el mensaje, ya que la condición del `if`, que es el valor devuelto por el método, es `true`.

boolean	`endsWith(String suffix)`	Devuelve `true` si el `String` al que se le envía el mensaje termina con la subcadena indicada en el argumento `suffix`.
		
		Ejemplo:
		`String nombre = "Diego Rodríguez Gracia";`
		`if (nombre.endsWith(Gracia)){`
		`  System.out.println("El 2º apellido es Gracia");`
		`}`
		Se escribirá el mensaje, ya que la condición del `if`, que es el valor devuelto por el método, es `true`.

int	`indexOf(String str)`	Devuelve la posición del `String` en la que se ha encontrado la primera ocurrencia del `String str` pasado como parámetro. Hay una variedad de métodos `indexOf()` adicionales, con funcionamientos ligeramente distintos, que puedes consultar en la API
		
		Un Ejemplo:
		`String nombre= "Juana María";`
		`int posicion=nombre.indexOf("ana");`
		`System.out.println("Posición = "+posicion);`
		
		Como resultado posicion toma el valor 2, que es el valor que se escribe.

TIPO DEVUELTO	NOMBRE Y SIGNATURA DEL MÉTODO	DESCRIPCIÓN DE SU FUNCIÓN.

int	`lastIndexOf(String str)`	Devuelve la posición del `String` en la que se ha encontrado la última ocurrencia (la ocurrencia más a la derecha) del substring `str` pasado como parámetro.
		
		Un Ejemplo:
		
		`String nombre= "Juana María";`
		`int posicion=nombre.lastIndexOf("a");`
		`System.out.println("Posición = "+posicion);`
		
		Como resultado posicion toma el valor 10, que es el valor que se escribe.

int	`length()`	MUY USADO: Devuelve un entero que es la longitud o número de caracteres que contiene el `String` para el que se ejecuta.
		
		Necesitamos usarlo frecuentemente cuando recorremos un `String` para realizar alguna manipulación, ya que de lo contrario intentaríamos acceder a posiciones más allá de la última, y se produciría un error, que haría abortar el programa.
		
		Ejemplo:
		
		Introducimos el valor del nombre del empleado desde teclado, y tenemos que asegurarnos de que realmente hemos tecleado el nombre, en vez de darle a la tecla `Intro` dejando el nombre vacío. Una posible forma de solucionarlo es la siguiente:
		
		`if (nombre.length()==0){`
		`  System.out.println("ERROR: Introducir el nombre es obligatorio.");`
		`}`

String	`replace(char oldChar, char newChar)`	Devuelve un `String` en el que se ha sustituido la primera ocurrencia del carácter `oldChar` por el nuevo carácter `newChar`, ambos pasados como parámetros en la llamada. Útil para corregir errores de un texto. También hay una variedad de métodos `replace()`. Es cuestión de seleccionar el que mejor se ajuste a las tareas que quieras realizar.

String	`substring(int beginIndex)`
	`substring(int beginIndex, int endIndex)`
		Devuelve el substring o subcadena que comienza en la posición `beginIndex` y que termina en la posición `endIndex-1`, ambas indicadas como parámetros. Si solo se da el comienzo del substring como parámetro, coge todos los caracteres desde esa posición hasta el final del `String`.
		
		Debemos tener cuidado de no salirnos de los límites del `String`, ya que si lo hacemos se produce una Excepción, un error que hace que aborte el programa. También hay varias modalidades de métodos `substring()`. Míralos en la API de Java
		
		Ejemplo:
		
		`String nombre= "Narciso Jáimez Toro";`
		`int posicionPrimerBlanco=nombre.indexOf(" ");`
		`int posicionSegundoBlanco=nombre.lastIndexOf(" ");`
		`String primerApellido=nombre.substring(posicionPrimerBlanco, posicionSegundoBlanco);`
		`String segundoApellido=nombre.substring(posicionSegundoBlanco);`

TIPO DEVUELTO	NOMBRE Y SIGNATURA DEL MÉTODO	DESCRIPCIÓN DE SU FUNCIÓN.

String	`toLowerCase()`	Sirve para pasar todo el `String` para el que se ejecuta el método a minúsculas.
		
		Ejemplo:
		`String nombre= "Gonzalo Fernández Hernández";`
		`System.out.println(nombre.toLowerCase());`
		
		Escribirá: `gonzalo fernández hernández`

String	`toUpperCase()`	MUY ÚTIL: Sirve para pasar todo el `String` para el que se ejecuta el método a mayúsculas.
		
		Por ejemplo, a la hora de guardar nombres que se han introducido desde teclado, suelen pasarse a mayúsculas, de forma que todos los nombres se almacenen en mayúsculas en los ficheros o bases de datos, con lo cual se simplifica la búsqueda de un nombre concreto, y se presentan todos los nombres de la misma forma. Y todo sin tener que obligar al usuario a introducirlos de una forma o de otra.
		
		Ejemplo:
		`String nombre= "Manuel Alberto Domínguez Vega";`
		`System.out.println(nombre.toUpperCase());`
		
		Escribirá: `MANUEL ALBERTO DOMÍNGUEZ VEGA`

String	`trim()`	Elimina los espacios en blanco que pueda contener el objeto `String` para el que se ejecuta, tanto al principio como al final.
		
		Útil para evitar que un nombre, por ejemplo, contenga esos caracteres, que no se ven, y que si se dejan provocarían que el nombre fuese más difícil de encontrar. Si introduzco un nombre de teclado, y le añado inadvertidamente espacios en blanco delante o detrás, o en ambos, tendré que corregirlo antes de almacenarlo.
		
		Ejemplo:
		`//nótese que contiene un espacio delante y otro detrás.`
		
		`String nombreAlmacenado= " Sebastián López Ojeda ";`
		
		`//en este caso, sin espacios delante ni detrás.`
		
		`String nombreAComparar= "Sebastián López Ojeda";`
		`if(nombreAlmacenado.equalsIgnoreCase(nombreAComparar)){`
		`  System.out.println("Encontrado");`
		`}else{`
		`  System.out.println("No se ha encontrado");`
		`}`
		
		Se escribirá como salida: `No se ha encontrado.`
		
		Sin embargo si usamos la condición …
		
		`if(nombreAlmacenado.trim().equalsIgnoreCase(nombreAComparar))`
		
		con los datos anteriores se escribiría: `Encontrado`

TIPO DEVUELTO	NOMBRE Y SIGNATURA DEL MÉTODO	DESCRIPCIÓN DE SU FUNCIÓN.

Static String	`valueOf(int i)`	Devuelve una representación como `String` de un número entero, en este caso. No obstante, hay versiones de este método para convertir casi cualquier tipo primitivo en un `String`, incluso para convertir en `String` un `Object`.
		
		Ejemplo:
		Queremos comprobar si un número entero empieza por la cifra 3:
		
		`int numero= 35454;`
		`String numeroCaracter = String.valueOf(numero);`
		`//Una vez convertido en String podemos usar los métodos`
		`//de manipulación de cadenas, para por ejemplo saber si //la primera cifra era un tres.`
		`if (numeroCaracter.charAt(0) = = '3'){`
		`  System.out.println(numero+"empieza por 3");`
		`}else {`
		`  System.out.println (numero+ "no empieza por 3" );`
		`}`
		
		La salida del código anterior es : `35454 empieza por 3`
		
		Fíjate en que es un método `static`. Eso quiere decir que no se ejecuta para un objeto `String` concreto, sino que hace algo para la clase, o dicho de otra manera: El mensaje de convertir un entero en `String`, no se lo mandamos a un objeto `String` concreto, si no a la clase `String`, que es la que sabe convertirlo. Y esto es así, porque yo puedo querer convertir valores `int` en `String`, aunque no haya creado ningún objeto `String`.
		
		Por el contrario, y a modo de ejemplo, no tiene sentido ejecutar un método de conversión a mayúsculas si no se tiene creado al menos el objeto `String` que se quiere pasar a mayúsculas. Por eso el método `toUpperCase()` no es `static`.