El concepto de Java new String es a veces difícil de entender para los programadores cuando uno empieza. Vamos a intentarlo explicar de forma sencilla . Cuando nosotros trabajamos con Strings en Java somos conscientes de que estamos trabajando con una clase que genera objetos. Por ejemplo podemos construir el siguiente código:

CURSO SPRING BOOT
GRATIS
APUNTATE!!

```
package com.arquitecturajava;

public class Principal {

	public static void main(String[] args) {

		String texto1= new String("hola");
		String texto2= new String("hola");
		System.out.println(texto1);
		System.out.println(texto2);
		System.out.println(texto1.equals(texto2));

	}

}
```

Acabamos de construir el ejemplo prácticamente de hola mundo e imprimir los valores por la consola.



No hay ninguna sorpresa ya que ambas cadenas son iguales. El resultado es true al usar el método equals . ¿Ahora bien qué diferencia existe entre ese bloque de código y el siguiente?.

```
package com.arquitecturajava;

public class Principal2 {

	public static void main(String[] args) {

		String texto1= "hola";

		String texto2="hola";

		System.out.println(texto1);

		System.out.println(texto2);

		System.out.println(texto1.equals(texto2));

	}

}
```

Java new String vs ==

En este caso no hemos usado el operador new simplemente hemos asignado directamente la literal. El resultado es idéntico a nivel de consola .



¿Entonces realmente da lo mismo? . La realidad es que ni de lejos da lo mismo . Si hacemos una pequeña modificación del código y utilizamos el operador == que nos compara los objetos a nivel de dirección de memoria.

```
package com.arquitecturajava;

public class Principal {

	public static void main(String[] args) {

		String texto1= new String("hola");

		String texto2= new String("hola");

		System.out.println(texto1);

		System.out.println(texto2);

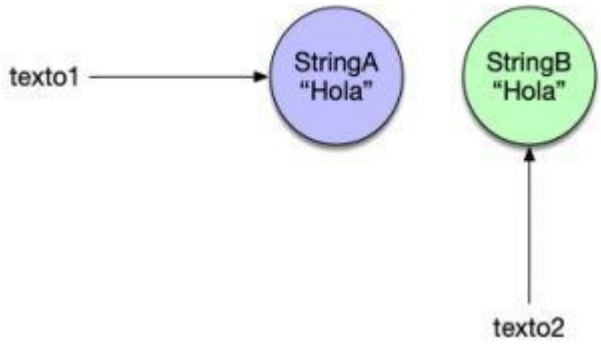
		System.out.println(texto1.equals(texto2));
		System.out.println(texto1==texto2);

	}
```

}

```
hola
hola
true
false
```

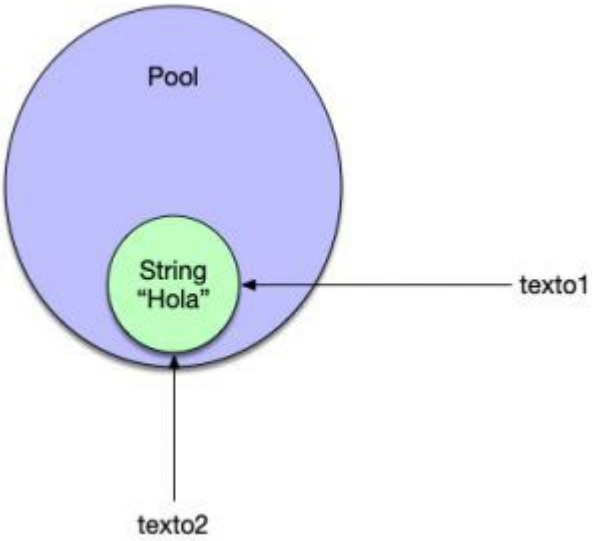
El resultado es clarificador aunque creemos la misma cadena en memoria , para Java son dos objetos diferentes .



Su igualdad es “true” ya que tienen el mismo texto de contenido , pero no son el mismo objeto. ¿Qué pasará en el caso de usar una asignación directa sin operador new? .Veamoslo.

```
hola
hola
true
true
```

En este caso solo se ha creado un objeto en memoria y las dos variables apuntan al mismo . Eso se debe a que Java a nivel de maquina virtual cada vez que generamos una literal las almacena en un pool para su posterior reutilización.



Esto en principio nos puede parecer poco importante pero si que lo es ,vamos a ejecutar un par de bucles for con las dos opciones:

```
package com.arquitecturajava;

public class Principal3 {

	public static void main(String[] args) {

		long tiempo = System.nanoTime();
		for (int i = 0; i &amp;lt; 1000000; i++) {

			String cadena = new String("hola");

		}

		long tiempo2 = System.nanoTime();

		for (int i = 0; i &amp;lt; 1000000; i++) {
```

```
                String cadena = "hola";

            }

            long tiempo3 = System.nanoTime();

            System.out.println(tiempo2- tiempo);
            System.out.println(tiempo3 - tiempo2);
        }

}
```

Estamos utilizando el método nanoTime de System que nos permite hacer un calculo exhaustivo del coste de cada uno de los dos bucles. Recordemos que el segundo bucle no genera nada mas que un objeto y lo mantiene en un pool por lo tanto debería ser más rápido:



Efectivamente la diferencia de rendimiento en un ejemplo tan sencillo como este ya es notable. Siempre que tengamos bloques de cadenas reutilizables es mejor asignar directamente una literal al String.

CURSO SPRING BOOT
GRATIS
APUNTATE!!

1. Java Stream String y Java 8
2. StringBuffer vs StringBuilder
3. Java BigInteger
4. Java Integer Wrapper y certificacion
5. Java String