String / StringBuilder / StringBuffer

String / StringBuilder / StringBuffer는 모두 문자열을 다루기 위한 클래스이지만 각 클래스의 동작 방식, 메모리 구조, 동기화 여부가 다름.

1. String(불변)

<특징>

- **불변성** (Immutable) : String 객체는 한 번 생성되면 그 내용을 변경할 수 없음. 어떤 문자열 연산을 수행하면 항상 새로운 String 객체가 생성됨

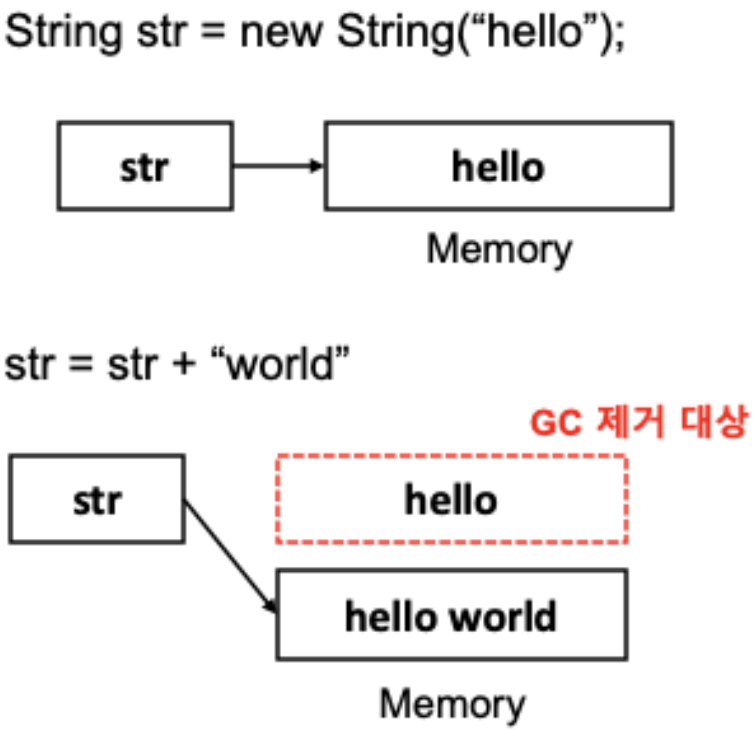
- **비동기화 :** String은 동기화를 제공하지 않음. 따라서 멀티 스레드 환경에서는 안전하지 않으나, 단일 스레드 환경에서 사용하기 적합함.

<메모리 구조>

String은 객체를 **new 연산자**를 사용하는 방법과 **문자열 리터럴**을 사용하는 방법 중 어떤 방식으로 생성하냐에 따라 메모리 구조에 차이가 있다.

new 연산자

```
String str = new String("hello");
str = str + "world";
```



new 연산자로 생성한 String 객체는 위 그림과 같이 메모리 구조가 형성됨.

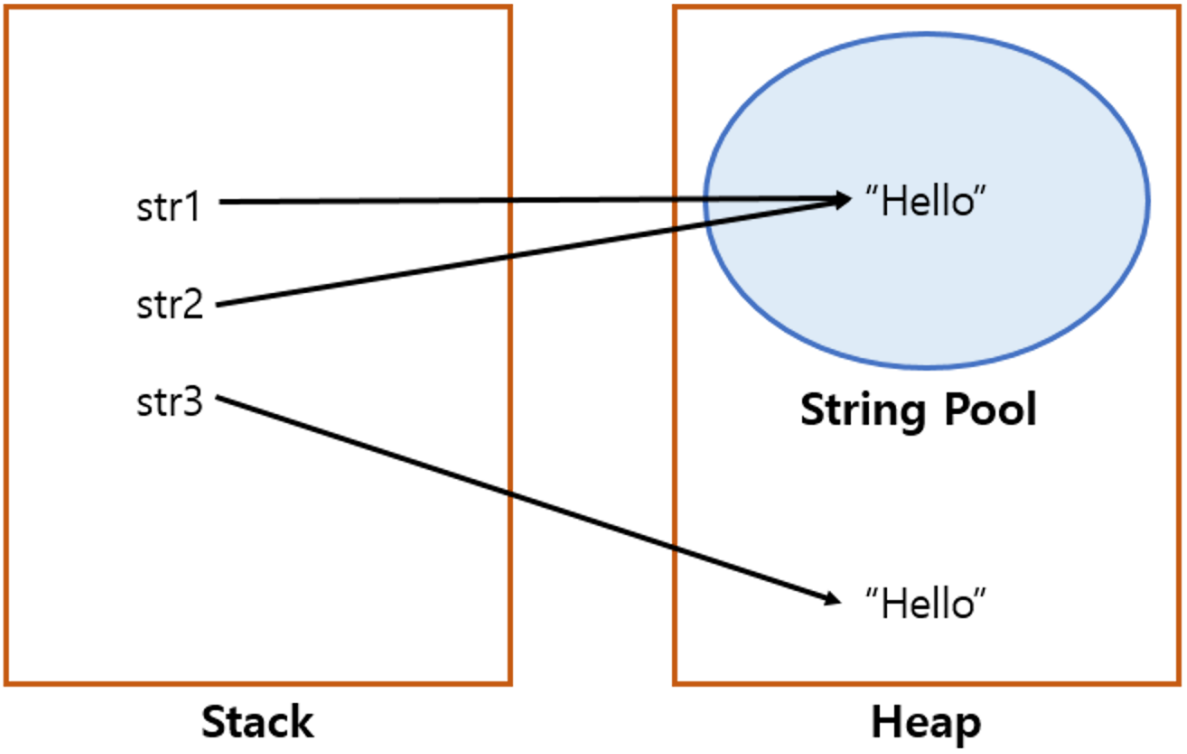
메모리의 Heap 영역에 새로운 String 객체를 매번 생성함.

문자열 리터럴

```
String str1 = "Hello";
String str2 = "Hello"; // str1과 같은 객체를 참조함 (String Constant P
ool에 의해)

String str3 = new String("Hello"); // Heap 영역에 새로운 객체를 생성함

System.out.println(System.identityHashCode(str1)); // 1509514333
System.out.println(System.identityHashCode(str2)); // 1509514333
System.out.println(System.identityHashCode(str3)); // 1552787810
```



문자열 리터럴은 JVM의 **"String Constant Pool(문자열 상수 풀)"**에 저장됨.

문자열 상수 풀은 메모리 내의 특별한 영역이며, 동일한 문자열 리터럴이 여러 번 사용될 경우 메모리 낭비를 방지하기 위해 이미 존재하는 문자열 객체를 재사용하게 함.

💡 `new` 연산자로 생성된 String 이든, 문자열 리터럴로 생성된 String 이든, String 의 **불변성은 변함이 없다.**

한 번 생성된 `String` 객체는 그 내용을 변경할 수 없다.

String 객체에 대해 어떤 변경 작업(ex. `concat`, `replace`, `toUpperCase` 등 문자열 관련 메서드)을 수행하더라도 원래 객체는 변경되지 않으며, 항상 새로운 String 객체가 생성된다.

2. StringBuilder(가변)

<특징>

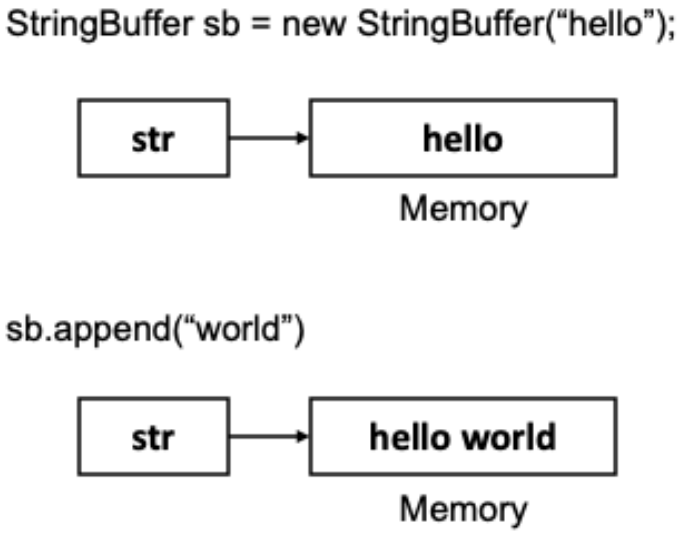
- **가변성 (Mutable):** StringBuilder는 문자열을 변경할 수 있다. 즉, 문자열 연산 시 새로운 객체를 생성하지 않고, 기존 객체의 내용을 변경한다.

- **비동기화 :** StringBuilder는 동기화를 제공하지 않음. 따라서 멀티스레드 환경에서는 안전하지 않으나, 단일스레드 환경에서 사용하기 적합함.

<메모리 구조>

StringBuilder는 내부적으로 char[] 배열을 사용하여 문자열을 저장하며, 이 배열은 필요에 따라 동적으로 크기가 늘어날 수 있다. (초기용량 16 : 문자 갯수)

배열 용량이 다 차면 자동으로 현재 크기의 2배로 확장됨.



3. StringBuffer(가변)

<특징>

- **가변성 (Mutable):** StringBuffer도 문자열을 변경할 수 있다. 즉, 문자열 연산 시 새로운 객체를 생성하지 않고, 기존 객체의 내용을 변경한다.

- **동기화 (Thread-Safe):** StringBuffer는 모든 메서드가 `synchronized` 키워드를 사용하여 동기화 되어 있다. 따라서 멀티스레드 환경에서 안전하지만, 동기화로 인한 성능 저하가 발생할 수 있다.

<메모리 구조>

StringBuffer의 메모리 구조는 StringBuilder와 동일하게 char[] 배열을 사용하며, 필요에 따라 크기가 증가함.

> 💡 결론적으로 중요한 사실은, StringBuffer와 StringBuilder 는 가변성을 띄기 때문에 기존 문자열에 추가적인 연산이 일어나도 새로운 객체를 생성하지 않고, 기존 객체의 내용을 변경한다는 점이다.

정리

특징	String	StringBuilder	StringBuffer
불변성	불변 (Immutable)	가변 (Mutable)	가변 (Mutable)
동기화	비동기화	비동기화	동기화 (Thread-Safe)
사용 환경	문자열이 변경되지 않을 때 사용	단일 스레드에서 문자열을 자주 변경할 때 사용	멀티스레드에서 문자열을 자주 변경할 때 사용
메모리 할당 방식	String Constant Pool + Heap	Heap (동적 배열 크기 조정)	Heap (동적 배열 크기 조정)
성능	느림 (변경 시 새로운 객체 생성)	빠름 (변경 가능)	느림 (동기화로 인한 *오버헤드)

📌 오버헤드(Overhead)란?

어떤 작업을 수행하는 데 있어서
직접적인 결과를 얻기 위해 필요한 추가적인 시간, 메모리, 또는 자원을 의미.