##### **Immutable**

String is a special ****class****. All Strings are object not primitive type.

String is an ****immutable class.****Means once created, the String's content cannot be updated, removed or inserted.

But we can do the following:

String s = "hi";

s = "hello";

s = s + " world";

It looks like the content of the string s got changed to "hello world" from "hi". But actually in the above code, four different String objects are created. s is just a reference. We can change the reference to point to any objects. But the String object itself never got changed.

so the below code is a bad practice:

String result = "";List<String> names = Arrays.asList("Adam","Bill","Charley");for( int i = 0 ; i < names.size(); i++){

retsult += names.get(i);}// result is a String combined each name in the list.// but in the for loop, new object got created in each iteration.// memory is wasted// Best practice is to use StringBuilder or StringBuffer

##### **String Pool**

Look at below code:

String s1 = "Hi";String s2 = "Hi";

s1 == s2 // will return true

String s3 = new String("Hi");String s4 = new String("Hi");

s3 == s4 // will return false

JVM maintains a String pool which stores all the String literals(char[]). The literals in String Pool can be reused. So when we declear a String object using quotes( s = "Hi"), it will reuse the "Hi" in the pool if it already exists. Since it is the same object, s1 and s2 are all pointing to the same object, so it gives true.

When using new keyword, the class won't check String pool, so it always creates new objects. so s3 == s4 is false.

##### **StringBuilder**

A mutable sequence of characters. This class provides an API compatible with StringBuffer, but with no guarantee of synchronization. This class is designed for use as a drop-in replacement for StringBuffer in places where the string buffer was being used by a single thread (as is generally the case). Where possible, it is recommended that this class be used in preference to StringBuffer as it will be faster under most implementations.

String result = "";

List<String> names = Arrays.asList("Adam","Bill","Charley");

for( int i = 0 ; i < names.size(); i++){

retsult += names.get(i);

}

// result is a String combined each name in the list.

// but in the for loop, new object got created in each iteration.

// memory is wasted, may cause extra GC time

// Best practice is to use StringBuilder or StringBuffer

Example:

StringBuilder sb = new StringBuilder();

for( int i = 0 ; i < names.size(); i++){

sb.append(names.get(i)); // only append new content to the same object.

}

result = sb.toString();

<https://www.youtube.com/watch?v=ffjsEQG8324>