**Strings**

In Java, Strings are **not** primitives - they are immutable, constant objects derived from the [String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html) class. To be immutable means that the state or value of the object cannot be altered once created - this is accomplished by having internal, private and final fields and not implementing any "setter" methods which would alter the state of those fields.

Because Strings are immutable, all of the methods in the String class return a **new** String - the original is not modified. For example:

String str1 = "my string";

str1.concat(" is the best!");

System.out.println(str1);

will print out my string. Why? Because the .concat() method **returns** a completely different string which we are not assigning to any variable, and the original object is not changed (it is immutable). Thus, str1 still refers to the String "my string". In order to make the code print "my string is the best!", we would need to change line 2: str1 = str1.concat(" is the best!"); which **re-assigns** the reference variable str1 to the new String returned from the method (the original String hasn't changed, however).

**String Pool**

When Strings are created they are placed in a special location within the heap called the String Pool. When String literals are created, if there is an existing String that matches in the String Pool, the reference variable will point to the existing value. Duplicates will not exist in the String Pool. This is important because Strings take up a lot of memory. Being able to reuse the same value throughout your application is advantageous.

One way to circumvent the above process is to use the new keyword along with the String constructor, which will explicitly create a new String object in memory, even if one already exists in the String Pool.

String a = "foo";

String b = "foo";

String c = new String("foo");

System.out.println(a == b); // true

System.out.println(a == c); // false