Java String Pool — **the special memory region where *Strings* are stored by the JVM**.

String Interning

the JVM can optimize the amount of memory allocated for them by **storing only one copy of each literal *String* in the pool**. This process is called *interning*.

When we create a *String* variable and assign a value to it, the JVM searches the pool for a *String* of equal value.

If found, the Java compiler will simply return a reference to its memory address, without allocating additional memory.

If not found, it'll be added to the pool (interned) and its reference will be returned.

Strings Allocated Using the Constructor

When we create a *String* via the *new* operator, the Java compiler will create a new object and store it in the heap space reserved for the JVM.

Every *String* created like this will point to a different memory region with its own address.

String Literal vs *String Object*

When we create a *String* object using the *new()* operator, it always creates a new object in heap memory. On the other hand, if we create an object using *String* literal syntax, it may return an existing object from the String pool, if it already exists. Otherwise, it will create a new String object and put in the string pool for future re-use.

At a high level, both are the *String* objects, but the main difference comes from the point that *new()* operator always creates a new *String* object. Also, when we create a *String* using literal – it is interned.

In general, **we should use the *String* literal notation when possible**. It is easier to read and it gives the compiler a chance to optimize our code.