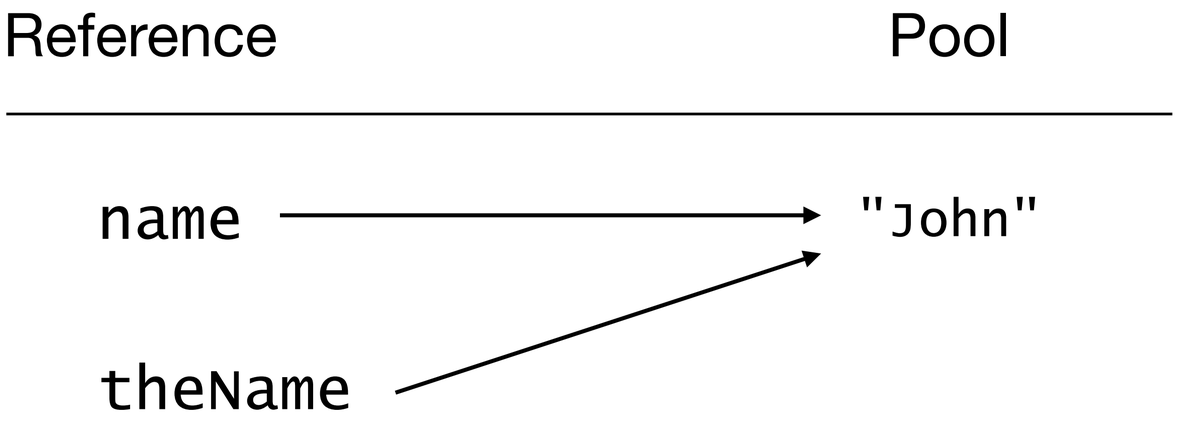
Strings

String Pool

What is String Pool?

- let's say you create a new string with literal value `"John"`
  - JVM stores it in the memory location known as **String pool** or **intern pool**
- now you create a new string variable and assign it a same literal value
  - instead of creating a new memory spot for this literal value
  - Java will save the memory and look in the String pool
  - new variable will point to the existing location in the String pool

```
String name = "John";

String theName = "John";
```



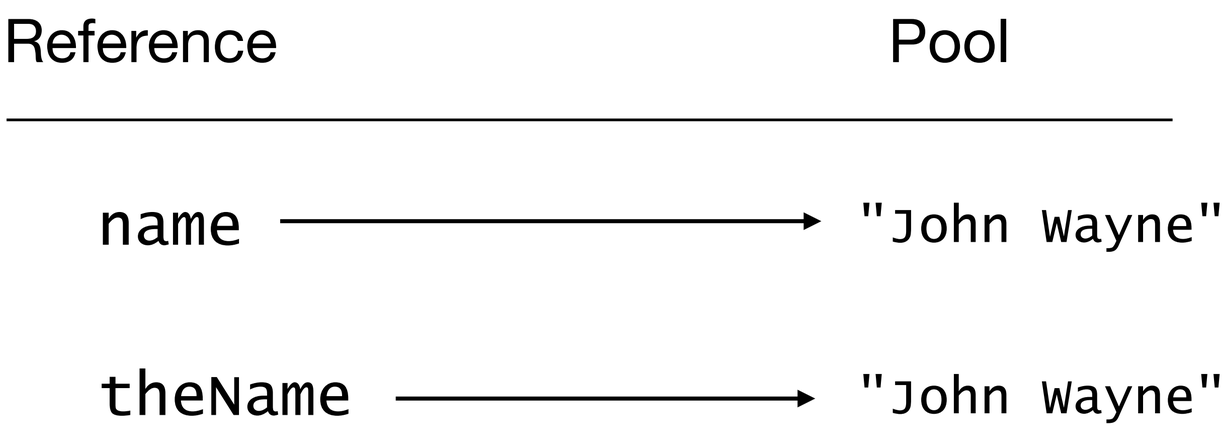

```
System.out.println(name == theName);

  => true   // both references point to the same memory location
```

```
// tricky example #1

String s1 = "John";

String s2 = "  John    ".trim();

System.out.println(s1 == s2);

  => false


// why?

// pool is created at compile-time, and trim() is evaluated at run-time
```

```
// tricky example #2

String s1 = "John Wayne"

String s2 = "John" + " " + "Wayne";

System.out.println(s1 == s2);

  => true


// concatenation is done in the compile-time

// compiler now knows that these are the same literals
```

```
// you can instruct the compiler to use the pool even with runtime methods
String s1 = "John";
String s2 = "  John    ".trim().intern();
System.out.println(s1 == s2);
  => true


// this also works (but it doesn't make much sense)
String s1 = "John";
String s2 = "  John    ".trim();
System.out.println(s1 == s2.intern());
  => true
```

```
// if you don't want the compiler to use the pool
// you can achieve this but creating a new object with keyword "new"
String name = "John Wayne";
String theName = new String("John Wayne");
```




```
System.out.println(name == theName);
  => false
```