Improved Equality Check

We'll cover the following

Types of equality checks

Just like Java, Kotlin also has two types of equality checks:

- `equals()` method in Java, or `==` operator in Kotlin, is a comparison of values, called structural equality.
- `==` operator in Java, or `===` in Kotlin, is a comparison of references, called referential equality. Referential equality compares references and returns true if the two references are identical—that is, they refer to the same exact instance. The operator `===` in Kotlin is a direct equivalent of the `==` operator in Java.

What is the difference?

But the structural equality operator `==` in Kotlin is more than the `equals()` method in Java. If you perform *str1.equals(str2)*; in Java, you may run into a `NullPointerException` if the reference *str1* is *null*. Not so when you use `==` in Kotlin. Kotlin's structural equality operator safely handles null references. Let's examine that with an example:

```
println("hi" == "hi")
println("hi" == "Hi")
println(null == "hi")
println("hi" == null)
println(null == null)
```

equality.kts

If these comparisons were done with `equals()` in Java, the net result would have been a runtime `NullPointerException`, but Kotlin handles the `nulls` safely. If the values held in the two references are equal then the result is `true`, and `false` otherwise. If one or the other reference is `null`, but not both, then the result is `false`. If both the references are `null`, then the result of the comparison is `true`. We can see this in the output, but you'll also see an added bonus in there:

```
true
false
false
false
true
equality.kts:3:9: warning: condition 'null == "hi"' is always 'false'
println(null == "hi")
        ^
equality.kts:4:9: warning: condition '"hi" == null' is always 'false'
println("hi" == null)
        ^
equality.kts:5:9: warning: condition 'null == null' is always 'true'
println(null == null)
        ^
```

The output confirms the behavior of `==` operator like mentioned. The output also shows yet another example of Kotlin's sensible warnings—if the result of comparison will always be an expected value, it prompts a warning suggesting we fix the code to remove the redundant conditional check.

When `==` is used in Kotlin, it performs the `null` checks and then calls `equals()` method on the object.

You've learned the difference between using `equals()` in Java and `==` in Kotlin.

In the next lesson, let's look at the ease with which we can create strings with embedded expressions.