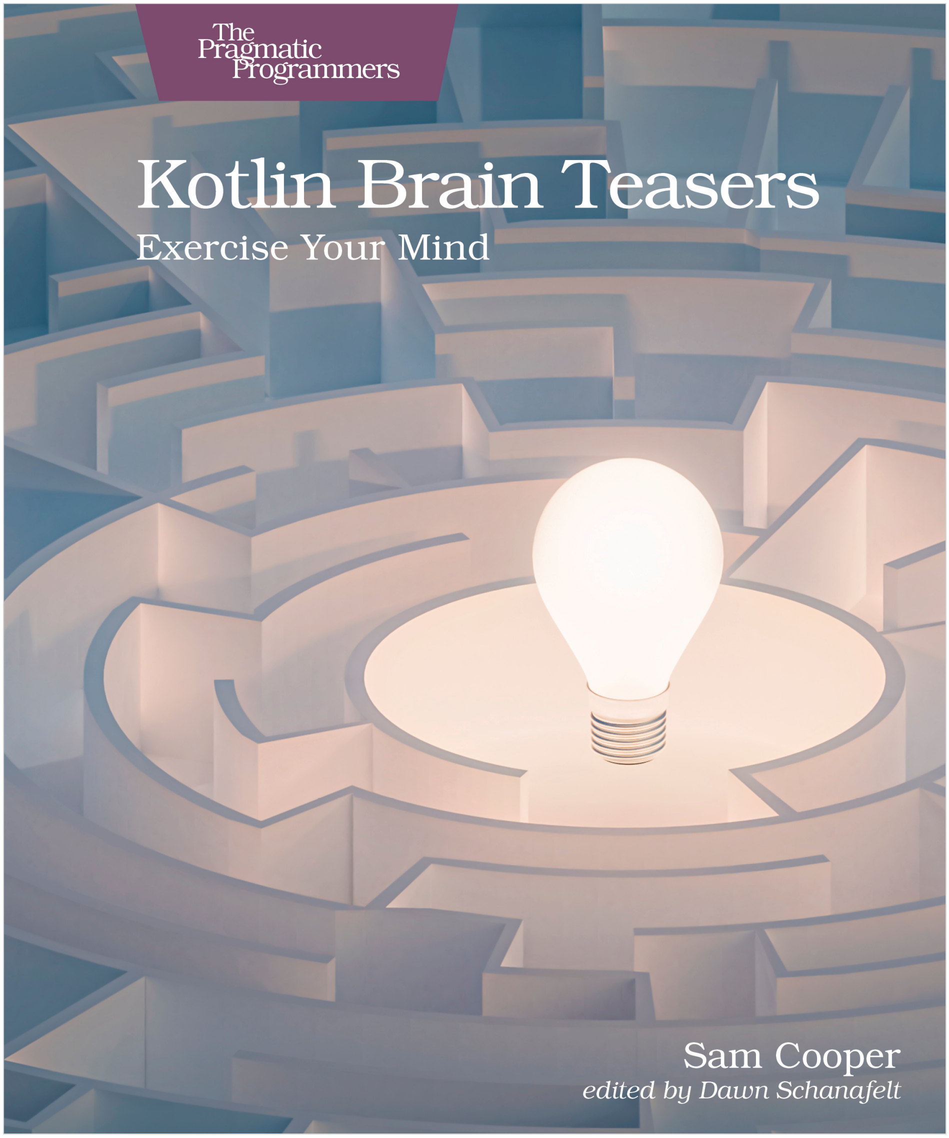
The Pragmatic Programmers
Kotlin Brain Teasers
Exercise Your Mind
Sam Cooper
edited by Dawn Schanafelt

Puzzle 1

Something From Nothing?

src/commonMain/kotlin/com/example/puzzles/operators/addnull/AddingNulls.kt

```kotlin
fun main() {
  val result = null + null

  if (result == null) {
    println("It's null")
  } else {
    println("It's a ${result::class.simpleName} with the value: $result")
  }
}
```

Guess the Output

Try to guess the output (or error) before moving to the next page.

The program displays the following output:

```
It's a String with the value: nullnull
```

Discussion

Why does adding null + null produce a String?

Kotlin’s *plus* operator (+) can do different things—adding numbers, merging lists, joining strings, and so on—depending on the types of its inputs. Of these possible operations, it turns out that string concatenation is the only one that can accept a null value on both sides. That leaves Kotlin with no choice but to interpret null + null as an attempt to combine two nullable strings.

Anytime you use null in a string-building operation like this, Kotlin will replace it with the literal text "null". We’ve provided two null inputs, so we get two copies of the word "null", which are then stitched together just like any other pair of strings would be.

Operator Overloads

This is a fun puzzle, because null + null looks like a meaningless and illogical expression that shouldn’t do anything at all—especially in a strongly typed language such as Kotlin. There’s certainly no reason to write it in a real program. So what can it teach us?

First, it’s a great demonstration of the way an operator can be *overloaded*. You probably already know how to overload a function—just make a second function with the same name, but with different input types. Well, operator overloads work in exactly the same way. In fact, Kotlin’s *plus* operator (+) is really a shorthand for calling a function named plus().

That means our null + null expression is the same as writing null.plus(null). To compile and execute it, Kotlin will search through all the plus() functions it can find, looking for one that can accept those values as inputs.

A null value will be accepted by any type that has a question mark (?) after it. For instance, the Any? type will accept null values, while the base type Any won’t. Every Kotlin type has a nullable counterpart like this, including basic types like strings and numbers. And sure enough, in the API documentation[3] you

3. https://kotlinlang.org/api/core/kotlin-stdlib/kotlin/plus.html

can find the overloaded plus() function we just used, complete with its null-accepting input types on both sides:

```
operator fun String?.plus(other: Any?): String
```

There are no other matching plus() functions with two nullable inputs, so the program selects this one automatically.

Member functions and extension functions can both be used as operator overloads. This particular version of plus() is an extension, declared outside the String class, which is what allows it to use a null receiver. Create your own operator functions, either as members of your own classes or as extensions on existing types, to make operators like *plus* (+) work with any inputs you like—nullable or not!

Displaying Nulls

The second thing this puzzle shows is how null interacts with text. A null value is just an unfilled slot in your program's memory. How do you transcribe the contents of a variable when that variable doesn't hold any data?

Instead of showing nothing, or worse, producing an error, Kotlin will represent a null value using the literal text "null":

```
val name: String? = null
println("Hello, $name!") // → "Hello, null!"
```

You'll see this automatic null-to-text conversion at work in a few different places, and it's important to be aware of it. When you're generating logs or diagnostic messages, it might be exactly what you need. But you probably don't want to display the word "null" to your end users, or worse, store it in a database field that was supposed to be blank.

That leaves us with two key takeaways from this puzzle:

- Remember that many operators are backed by overloadable functions. This will help you understand and even extend their behavior.
- When you're building strings, don't forget to consider how you're handling null values, and whether the literal word "null" is an appropriate replacement.

Further Reading

Operator overloading
  https://kotlinlang.org/docs/operator-overloading.html

Null safety
  https://kotlinlang.org/docs/null-safety.html