vararg and Spread

We'll cover the following

Functions like `println()` take a variable number of arguments. The `vararg` feature of Kotlin provides a type-safe way to create functions that can receive a variable number of arguments. The spread operator is useful to explode or spread values in a collection as discrete arguments. We'll look at `vararg` first and spread next.

Variable number of arguments

In Functions with Block Body, we wrote a `max()` function that took an array of numbers. In the call to the function, as expected, we passed an array of values. If we already have an array of values, then it's not a big deal; but if we have a discrete set of values, then to call the function we'll have to create a temporary array of those values and then pass that array. Tedious.

In Kotlin, functions may take a variable number of arguments. Let's convert the `max()` function to be more flexible for the caller.

```kotlin
fun max(vararg numbers: Int): Int {
  var large = Int.MIN_VALUE

  for (number in numbers) {
    large = if (number > large) number else large
  }

  return large
}
```

varag.kts

Compared to the `max()` function we wrote previously, this version has two changes, both in the parameter list. First, the parameter `numbers` is prefixed with the keyword `vararg`. Second, the type of the parameter is specified as `Int` instead of `IntArray`. The real type of the parameter `numbers` is an array; the `vararg` annotates the parameter to be an array of the specified type.

Let's make a couple of calls to the function, passing a few discrete values.

```
println(max(1, 5, 2)) //5
println(max(1, 5, 2, 12, 7, 3)) //12
```

vararg.kts

That worked nicely—we can pass any number of arguments and Kotlin's type checking will ensure that the different arguments are of the right type.

The `max()` function took only one parameter, but you can use `vararg` when a function takes more than one parameter as well. But only one parameter may be annotated as `vararg`.

Here's a function that takes two parameters, but only the trailing one is marked as `vararg`.

```
fun greetMany(msg: String, vararg names: String) {
  println("$msg ${names.joinToString(", ")}")
}

greetMany("Hello", "Tom", "Jerry", "Spike") //Hello Tom, Jerry, Spike
```

mixvararg.kts

In the call, the first argument binds to the first parameter, and the remaining arguments are passed to the `vararg` parameter.

The type of the `vararg` parameter may be independent of the type of any of the other parameters the function takes.

The `vararg` parameter isn't required to be the trailing parameter, but I highly recommend that it is. Consider the following version of the `greetMany()` function:

```
fun greetMany(vararg names: String, msg: String) {
  println("$msg ${names.joinToString(", ")}")
}
```

When calling the function, if you pass any number of unnamed `String` arguments the compiler will assume all of them are for the `vararg` parameter. For it to know that a value is for the `msg` parameter, you'll have to use a named argument, like so:

```
greetMany("Tom", "Jerry", "Spike", msg = "Hello") //Hello Tom, Jerry, Spike
```

If you annotate a non-trailing parameter with `vararg`, then the caller is forced to use named arguments.

Here are some recommendations on where to place the `vararg` parameter:

- Place it in trailing position so callers aren't forced to use named arguments.
- Place it before the last parameter if the last argument is expected to be a lambda expression—we'll explore this further later in the course.

We saw how Kotlin makes it easy to pass a variable number of arguments, but what if we have an array with values already? The spread operator comes to the rescue in that case.

The spread operator

Take another look at the `max()` function with `vararg` in the previous example—it made it easier to pass different numbers of arguments. Sometimes, though, we may want to pass values that are in an array or list to a function with `vararg`. Even though the function may take a variable number of arguments, we can't directly send an array or a list. This is where the spread operator comes in. To look at an example of using this operator, let's start with the following instance:

```
// vararg.kts
val values = intArrayOf(1, 21, 3)
```

The `vararg` implies that we may pass any number of arguments to this single parameter. But if we pass an array as argument, we'll get an error:

```
println(max(values)) //ERROR
//type mismatch: inferred type is IntArray but Int was expected
```

Even though internally the type of `vararg` parameter is an array, Kotlin doesn't want us to pass an array argument. It'll only accept multiple arguments of the

specified `vararg` type.

To use the data in the array, we may try the following:

```
// vararg.kts
println(max(values[0], values[1], values[2])) //SMELLY, don't
```

But that's verbose, and you'll never be able to proudly show that code to anyone, not even to mom.

Where the parameter is annotated as `vararg`, you may pass an array—of right type, of course—by prefixing it with the spread operator `*`. Let's pass the array values as an argument to the numbers parameter of `max()`:

```
// vararg.kts
println(max(*values)) //21
```

That's much better. By placing a `*` in front of the argument, you're asking Kotlin to spread the values in that array as discrete values for the `vararg` parameter. No need to write verbose code; the combination of `vararg` and spread restores harmony.

If we have an array, we can use spread, but often we work with lists instead of arrays. If we want to pass a list of values, then we can't use the spread operator on the list directly. Instead, we have to convert the list to an array, of the desired type, and then use spread. Here's an example of how that would look:

```
// vararg.kts
println(max(*listOf(1, 4, 18, 12).toIntArray()))
```

If the elements in the list and the type of `vararg` are of a different type than `Int`, then use the appropriate `to...Array()` method of `List<T>` to convert to an array of the desired type.

QUIZ

1 What is `varag` used for?

2 How can we annotate a non-trailing parameter with `varag`?

3 Which of the following is the spread operator symbol?

Retake Quiz

In the next lesson, we’ll learn about destructuring in Kotlin.