Function Fluency with infix

We'll cover the following

The infix notation

Dots and parenthesis are common in code we write, but on many occasions leaving those out can make the code less noisy and easier to follow. For example, the following is familiar code in Java:

```
//Java
if(obj instanceof String) {
```

Imagine Java had insisted that we write `if(obj.instanceOf(String)) {`—what a clutter that would have been. Instead, we used a nice syntax `if(obj instanceof String) {`—that's much easier to read, and it uses what is called the infix notation, where an operator is infixed or implanted in the middle of its operands. That syntax is nice, but in Java such fluency is limited to predefined operators. In Kotlin you can use the infix notation—that is, leave out dots and parenthesis—for your own code and thus make the code expressive, less noisy, and easier to read.

In the earlier example with Circle and Point, we saw this:

```
println(circle.contains(point1)) //true
println(point1 in circle) //true
```

We used the dot and parenthesis to invoke `contains()` but didn't use those with `in`. The reason is obvious: `contains()` is a method, but `in` is an operator, like `+`. Operators always enjoy infix notation automatically in Kotlin, but methods don't by default. That's a good reason for the difference, but we don't have to accept that and settle quietly. If we want to use `contains` much like the way we used `in`, we can do that in Kotlin, but after a change.

Using `infix`

If we mark a method with `infix` annotation, then Kotlin will allow us to drop the dot and parenthesis. `infix` may be combined with `operator`, but `operator` isn't required for `infix`—they are orthogonal to each other.

To see `infix` notation in action, we'll start with this:

```
operator fun Circle.contains(point: Point) =
  (point.x - cx) * (point.x - cx) + (point.y - cy) * (point.y - cy) <
    radius * radius
```

Let's change it to the following:

```
operator infix fun Circle.contains(point: Point) =
  (point.x - cx) * (point.x - cx) + (point.y - cy) * (point.y - cy) <
    radius * radius
```

Now, we can write:

```
println(circle.contains(point1)) //true
```

And we can also write:

```
println(circle contains point1) //true
```

What's the big deal in not having dot and parenthesis? you may ask. It leads to more fluent code and less noise—characteristics that are helpful for creating DSLs—we'll see this in Chapter 14, Creating Internal DSLs.

Kotlin offers some flexibility for `infix` functions but also comes with some limitations: `infix` methods are required to take exactly one parameter—no `vararg` and no default parameters.

In the next lesson, we'll use four methods to write more fluent code.