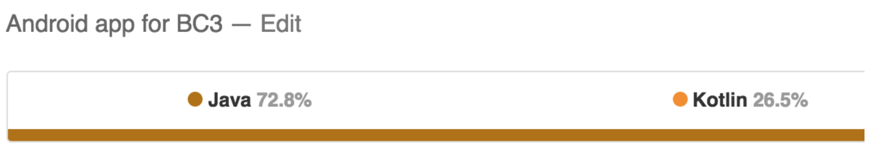
Dan Kim Follow
Android Programmer at Basecamp | https://twitter.com/dankim
May 28, 2016 · 5 min read

Some of my favorite Kotlin features (that we use a lot in Basecamp)

· · ·

Team Android at Basecamp recently passed a fairly big milestone—over 25% of the Basecamp 3 Android app code base now runs on Kotlin! 🎉


Android app for BC3 — Edit

● Java 72.8% ● Kotlin 26.5%

Github statistics for the Basecamp 3 Android app as of 5/27/16.

We've found that Kotlin not only makes our code much better, but massively increases programmer happiness. All of this ensures we're making the best app we can for the tens of thousands of Android users we support.

Given our new experiences with the language, I thought it'd be worth sharing some specifics that make the language so wonderful to work with.

· · ·

Unlike most articles that introduce you to a language, I'm going to avoid using too much programming lingo. Instead, I'll try using plain English in the hopes that it's more accessible to beginners. 🙌

Some notes about the code examples:

- I am by no stretch an expert in Kotlin. Read, consider, and discuss!
- They look better on a desktop browser. You can get by on the mobile app in landscape mode, but I'd recommend breaking out your laptop to read them.

- They're brief and simple on purpose. Long-winded examples tend to cause confusion. Take these simple examples and extrapolate them into your own potential uses, and you'll see a lot more power.

Let's get started with seven of my current favorites!

. . .

1. Replacing simple if/else if/else blocks with when

One of my absolute favorites.

```
// Java
if (firstName.equals("Dan")) {
    person.setTeam(programmers);
} else if (lastName.equals("Dihiansan")) {
    person.setTeam(designers);
} else {
    person.setTeam(others);
}


// Kotlin
when {
    firstName == "Dan"       -> person.team = programmers
    lastName  == "Dihiansan" -> person.team = designers
    else                     -> person.team = others
}
```

when blocks are effectively the same as a simple *if* block, but look how much more readable that is!

There's a similar convention when only one argument is being checked. Typically this would be a long, ugly switch/case statement in Java.

```
// Java
switch (firstName) {
    case "Dan": person.setTeam(programmers)
        break;
    case "Jay": person.setTeam(programmers)
        break;
    case "Jamie": person.setTeam(designers)
        break;
    default:
        person.setTeam(others)
}


// Kotlin
when (firstName) {
    "Dan", "Jay" -> person.team = programmers
    "Jamie"      -> person.team = designers
    else         -> person.team = others
}
```

I swear, this alone is worth writing Kotlin.

2. Beautifying even the ugliest click handlers

Using Anko, a library built for Kotlin, click listeners are ridiculously easy.

I hate writing these in Java so much that I could barely bring myself to write an example here. But I soldiered on. 😭

```
// Java
view.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        System.out.println("This is horrible");
    }
});


// Kotlin
view.onClick {
    println("WAT")
}
```

3. No more view binding

By using the Kotlin Android Extensions, you no longer need to bind views to objects to start working with them. You can access them directly without any binding. Zero. None.

```
// Java
EditText composer = findViewById(R.id.composer);
composer.setText("Allo!");


// Kotlin
view.composer.text = "Allo!"
```

That might not look like a big deal in isolation, but think about how much of your Activity/Controller code is the ceremony of binding a view to an object before you can start to work with that object. Kotlin bypasses all of that.

4. Functions in one line

One line functions can technically be written in Java, but you'd be going against generally accepted styles.

Kotlin's inherent brevity makes one-liners (officially called single-expression functions) quite common, and they look great. No extra lines and no braces required.

```
// Java
public String fullName() {
    return getFirstName() + " " + getLastName();
}

// Kotlin
fun fullName() = "${firstName} ${lastName}"
```

Bonus: the return object type is implied, so Kotlin will automatically know the method is returning a String without ever having to write "String" anywhere.

You may have also noticed in this example 1) no need for *public* and 2) string interpolation.

5. Convenience methods built on top of familiar objects

Kotlin has extended objects you're familiar with and made them even better and packaged them into the Kotlin Standard Library.

Take String comparisons for example:

```
// Java
if (name.toLowerCase().contains(firstName.toLowerCase())) {
    ...
}

// Kotlin
if (name.contains(firstName, true)) { ... }
```

Not a huge difference, but enough to improve
readability in many places. The standard library has tons of these kinds of tweaks. Perfect!

6. Reducing the need for `if (whatever != null)`

Null checking is so painfully common in Java that `if (whatever != null)` is probably in your recurring nightmares.

Kotlin has a number of impressive null safety features built in, and *let* is just one of those ways to achieve more readable code.

```
// Java
if (message != null) {
    System.out.println(message)
}

// Kotlin
message?.let { println(it) }
```

Here if *message* is not null, Kotlin will *let* the block (what's inside the braces) run. If it's null, it just skips it.

There's one other bit of awesomeness—notice the *println(it)* statement? The *it* keyword allows you to reference the object the *let* began from.

7. The Elvis operator

I mostly love this operator because of its name. It looks like this:

```
?: // Turn your head to the left, you may see someone
familiar
```

Fun name aside, the real reason this is great is because it handles the common scenario of "if something is null, I want to give it a value, but otherwise just leave it alone".

```
// Java
if (people == null) {
    people = new ArrayList();
}
return people;

// Kotlin
return people ?: emptyArrayList()
```

Here if *people* isn't null, it returns. If it is, it returns whatever is to the right of the Elvis operator.

. . .

So that's just a brief look at some things that make my life better every day working with Kotlin.

If you're interested in getting started with Kotlin, their documentation is *very* good, and you can poke around the interactive Kotlin Koans. I tend to struggle with things like koans (feels too much like school work!), so if you're like me, I'd encourage you to try building something real.

. . .

We're working really hard to make the all-new Basecamp 3 and its companion Android app as great as they can be. Check 'em out!

If this was helpful to you, please do hit the heart button below or let me know on Twitter and we'll keep adding to this Kotlin series.