Table of Contents

How do I write Kotlin?

Language idioms

Kotlin was originally developed as a JVM language, and has thus been considerably influenced by the Java world. While this means that some fragments of code may be translated line-by-line into Kotlin, this often does not result in optimal code.

Take the following example:

```
final var person = ...;

if (person != null) {
    final var addresses = person.getAddresses();
    if (addresses != null && !addresses.isEmpty()) {
        final var firstAddress = addresses.get(0);
        if (firstAddress != null) {
            final var street = firstAddress.getStreet();
            if (street != null) {
                final var streetName = street.getName();
                if (streetName != null) {
                    ...
                }
            }
        }
    }
}
```

One might argue that this Java antipattern is obsolete, due to the advent of Java 8 and its Optional utility class. Let us give this sample a more modern twist:

```
final var person = ...;

Optional.ofNullable(person)
        .flatMap(Person::getAddresses)
        .flatMap(as -> as.stream().findFirst())
        .flatMap(Address::getStreet)
        .flatMap(Street::getName)
        .ifPresent(streetName -> {
            ...
        });
```

While this is much more concise, consider the overhead of wrapping every value that *may or may not* have a value with an extra object that:

- must be allocated on the stack (or heap)
- incurs a runtime performance penalty for each of the chained method calls
- results in code that is non-trivial to read

Not only is this be an unnecessarily complex approach to handling Tony Hoare's billion-dollar mistake, there exist *core language features* in Kotlin designed to avoid this problem entirely:

```
val person = ...

person?.addresses?.firstOrNull()?.street?.name?.let {
    ...
}
```

In short: to write proper, idiomatic Kotlin, you will *need* to ignore your instincts and find new approaches to solving problems. Pretending to write Kotlin when you are doing nothing more than translating the Java you *would've* written will not result in good code.

IntelliJ's built-in Java-to-Kotlin converter (*Ctrl+Alt+Shift+K*) is often a great start; its conversions are relatively decent due to the IDE's advanced static analysis capabilities. This is an invaluable tool for learning the language; it will recognize common patterns and translate them into Kotlin appropriately. Of course, one should not rely solely on this tool — it will sometimes produce suboptimal (or even invalid) code, though it is constantly being improved.