Kotlin Overview: Principles and Goals

Find out Kotlin's guiding principles, how they affect language design, and why this is relevant for developers.

WE'LL COVER THE FOLLOWING

Kotlin’s language design is guided by its underlying principles and goals.

Principles and Goals

Kotlin aims to be a readable, pragmatic, safe, and interoperable programming language:

- **Readability** is supported by language features such as type inference, data classes, and infix functions. Such features allow writing concise code without losing readability.
- **Pragmatism** is crucial because Kotlin is for large-scale enterprise software development. JetBrains use it themselves to develop their IDEs. Thus, Kotlin incorporates industry feedback and addresses issues of large-scale software development.
- **Safety** aims to prevent common software bugs by design. This is aided by several language features such as nullable types (to prevent null pointer exceptions) and by nudging you towards best practices such as designing for inheritance.
- **Interoperability** with Java is a major selling point of Kotlin and a necessary base for its widespread adoption in the JVM world. Interoperability allows Kotlin and Java to be used side by side, including the use of Java libraries or frameworks from Kotlin. For instance, the Kotlin standard library interoperates with Java by reusing the Java

Collections API. Similarly, it interoperates with JavaScript in the context of Kotlin/JS.

> **Note:** The concept of interoperability with Java will be used in the code examples in this course. For instance, we will use types from the Java standard library, such as `Date`.

Implications for Developers

Kotlin's principles guide its language design decisions and, as a result, the way you develop in Kotlin.

- Kotlin makes you think about nullability from the start by integrating nullability into the type system.
- Kotlin is intentionally constrained in certain areas (e.g. operators) to aid readability.
- Kotlin favors immutability and functional programming, for instance by nudging you to use read-only (`final`) variables.
- Kotlin drops checked exceptions to aid scalability in large-scale software development.
- Since it was developed by a tooling company, Kotlin focuses on toolability. For example, variable types are defined after variable names to remove ambiguity while parsing for the compiler (but also due to type inference and other reasons).

Summary

In short, it's important to know the philosophy of a language to understand why some things work the way they do. Ideally, your personal development philosophy matches with that of the language you use.

Next up is a brief lesson with some tips on how you can make the most of this course.