Introduction to Kotlin

Christian Konersmann, Finn Paul Lippok, Paul Lukas

RWTH Aachen University, Germany
{christian.konersmann,finn.lippok,paul.lukas}@rwth-aachen.de

March 21, 2025

Abstract

This paper is an introduction to Kotlin, a statically typed, object-oriented programming language designed to be fully interoperable with Java and the Java Virtual Machine (JVM). Kotlin offers a concise syntax, functional programming paradigms, and safety improvements compared to Java. In 2019, Google announced that Kotlin replaced Java as their preferred language for Android development.

1 Introduction

Introduction, motivation, and goals of this paper. This paper assumes that the reader is familiar with the fundamentals of Java. This paper was written as part of the *Proseminar: Advanced Programming Concepts*.

2 Basic Syntax

This section will cover the basic syntax of Kotlin and highlight the changes compared to Java. The goal of this section is to provide a brief overview, focusing on the most important differences.

2.1 Main Method

The main method is the entry point of every Java and Kotlin program. Java enforces object-oriented programming, thus requiring the main method to be declared inside a class. For the main method to be directly executable, the method must be declared as static and public.

Java main method

```java
public class Main {
  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }
}
```

Kotlin, on the other hand, does not require methods to be declared inside a class, allowing for a more functional programming style with top-level functions. These top-level functions can be called directly without the need to create an instance of a class, similar to static methods in Java[1] but without class affiliation. Kotlin further reduces boilerplate code by changing the default visibility to public and allowing the main method to be declared without arguments passed as an array. Some further basic syntactical changes include making the semicolon optional and introducing the *fun* keyword for defining functions. These changes lead to a more concise and readable main method and syntax in general.

Kotlin main method

```kotlin
fun main() {
  println("Hello, World!")
}
```

2.2 Type Declaration

In Kotlin, a variable declaration starts with the keyword *val* for immutable variables or *var* for mutable variables, similar to Java's *final* and non-final variables. The type of a variable is declared after the variable name, separated by a colon.

Java data types

```java
    final String name = "John Doe";
    int age = 42;
```

Kotlin data types

```kotlin
    val name: String = "John Doe"
    var age: Int = 42
```

2.3 Type Inference

Kotlin also supports type inference, allowing the compiler to infer the type of a variable based on its initializer.

```kotlin
    val name = "John Doe" // type is inferred as String
    var age = 42 // type is inferred as Int
```

2.4 Method Declaration

Similar to java. void = Unit which is optional.

Java method declaration

```java
    public int add(int a, int b) {
      return a + b;
    }
```

Kotlin method declaration

```kotlin
    fun add(a: Int, b: Int): Int {
      return a + b
    }
```

[1] When compiling Kotlin to Java bytecode, top-level functions are compiled to static methods in a class named after the file name.

2.5 Everything is an Object

In Kotlin, everything is an object, including primitive types and functions.

3 New Language Constructs

This section focuses on the most important new language constructs that are not present in Java. This section will illustrate Kotlin's advantages using a list of salespersons as an example and comparing it to Java.

3.1 Classes

3.2 Properties

3.3 String Interpolation

3.4 Null Safety

3.4.1 Smart Casts

3.4.2 Null Safety Operators

4 Interoperability

5 Android

6 TODO

6.1 General

Come up with an example that shows the differences between Java and Kotlin and highlights the advantages of Kotlin.

6.2 Introduction

Android development, improvements over, and interoperability with Java. Introduce an example to show differences/translation between Java and Kotlin.

6.3 Basic Syntax

- Methods
- Everything is an object (no primitives, functions are objects)
- Explain the absence of static methods (out of scope for introduction?) (use `@JvmStatic` annotation for interoperability)

6.4 Interoperability

- Explain the interoperability between Java and Kotlin (e.g. Using Java libraries in Kotlin)
- Use of annotations (e.g. `@JvmStatic`, `@JvmField`, `@JvmName`, `@JvmOverloads`) (out of scope for introduction?)

- Compile to other languages (e.g. JavaScript, Native) (out of scope for introduction?)

6.5 New Features

- Null safety
- Properties (Getters, Setters)
- Extension functions (not as important)
- *if* and *when* as expressions (not as important, only if it fits)

6.6 Android

- Discuss Kotlin's advantages for Android development

Acknowledgements

We would like to thank our instructor.