Pragya Kansal, Jesutofunmi Adediji

Background

- Kotlin was created by JetBrains -> Wanted to create a language similar to Java that is more concise and has higher-order functions
- Typically used for Android development
- Can also compile to JavaScript, meaning it can be used to write client-side web applications
- Has both functional and object-oriented constructs
- Reduced development time in comparison to using Java
- Statically typed language, meaning that type doesn't have to be explicitly declared
- Completely interoperable with Java code
- Introduced smart casting and coroutines
- Provides null safety → ensures that null values can't be accessed or manipulated

Code Sample

```
fun sayHello(greeting: String, itemToGreet:String) = println ("$greeting $itemToGreet")

fun calculateAverage(numbers: List<Double>): Double {
    if (numbers.isEmpty()) {
        return 0.0 // Handle the case when the list is empty to avoid division by zero.
    }

    val sum = numbers.sum()
    return sum / numbers.size
}


fun main() {
    val names= arrayOf("Pragya", "Tofunmi", "Prof. Laufer")
    names.forEach { name ->
        sayHello("Hi", name)
    }
    println()
    println("**************************")
    val numbers = listOf(5.0, 10.0, 15.0, 20.0, 25.0)
    val average = calculateAverage(numbers)
    println("Average: $average")

}
```

```
Hi Pragya
Hi Tofunmi
Hi Prof. Laufer

**************************
Average: 15.0
```

Live Code Sample

Classification - Paradigms

- Object-oriented
  - Uses classes, objects, inheritance, encapsulation, and polymorphism
- Functional
  - Supports lambda expressions, higher-order functions, and function types
  - Promotes immutability and higher-order operations on collections
- Concurrent and Asynchronous
  - Easy to write asynchronous code using coroutines
- Imperative
  - Involve sequences of statements that alter a program's state
  - Mutable variables using 'var'
  - Loops and conditional statements
- Declarative
  - Declarative code usually used when working with collections and data transformations
  - Immutable variables using 'val'

Classification - Design Principles

- Abstraction: it allows defining classes, functions, and interfaces to factor out recurring patterns in your code. It supports various abstraction mechanisms, such as classes, interfaces, and functions.
- Automation: Kotlin helps automate various tasks with its concise syntax and features like type inference, extension functions, and smart casts, which reduce the need for boilerplate code and make coding less error-prone.
- Defense in Depth: Kotlin allows you to implement multiple layers of security and validation within your code, making it possible to catch errors at various levels of the application.
- Information Hiding: Kotlin supports encapsulation and information hiding through features like access modifiers (private, protected, internal) and properties, allowing you to expose only what's necessary for the user and implementor.
- Labeling: Kotlin supports named arguments and data classes, which make it easier to work with labeled data and avoid the need to know the absolute position of items in a data structure.
- Manifest Interface: it's syntax and language features make interfaces and their contracts apparent in the code, helping developers understand how to use them correctly.

Continuation

- **Orthogonality**: Kotlin provides independent mechanisms for various language features, enabling you to control different aspects of your code independently.
- **Portability**: Kotlin is designed to be a cross-platform language and is not tied to a specific machine or class of machines, making it portable across different platforms and systems.
- **Preservation of Information**: Kotlin allows you to represent information in various forms, including data classes, sealed classes, and user-defined types, preserving the information you need.
- **Regularity**:Kotlin follows a regular and consistent syntax, making it easier to learn, use, and implement. It avoids many irregularities present in some other languages.
- **Security**: Kotlin helps catch many programming errors at compile time, enhancing security. It enforces type safety and null safety, reducing the risk of runtime errors.
- **Simplicity**: Kotlin aims to be a simple and concise language, with a minimal number of concepts and features. It prioritizes readability and ease of use.
- **Structure:** Kotlin's static structure typically corresponds well to the dynamic structure of computations, making code organization more straightforward.
- **Syntactic Consistency**: Kotlin promotes consistency in its syntax, making similar things look similar and different things distinct.
- **Zero-One-Infinity:** Kotlin provides support for zero, one, and multiple instances of objects, collections, and other data structures, aligning with this principle.

Evaluation

- Readability
  - Has less boilerplate code than Java, making the codebase cleaner
- Writability/Productivity
  - More concise and expressive syntax than Java → 40% less code
  - Higher learning curve for those without experience in Java
- Reliability
  - Distinction between nullable and non-nullable types reduces likelihood of null-pointer exceptions
  - Type-related errors caught at compile time → prevents type mismatches
  - Has a wide range of functions that are well-tested
  - Works with Java libraries and frameworks, making transition to Kotlin simple and low-risk

Evaluation

- Cost
  - Open source and free to use, reducing development costs
  - May be upfront costs due to migrating code to Kotlin
- Portability
  - Can be used to write code that works with multiple different platforms
  - Can be used for web, server-side, Android, and native development
  - Supported by various different IDEs
- Community, industry backing
  - Adopted by several major companies and has several online resources
  - Used in many open source projects in the Android development community
- Major projects
  - Netflix, Uber, Trello, Pinterest, JetBrains products
- Ecosystem
  - Android Studio, IntelliJ Idea → features such as code completion, debugging, and refactoring

Conclusion

Advantages:

- Conciseness
- Null safety
- Interoperability: allows you to use existing java libraries and framework at ease.

Disadvantages:

- Compilation Time

Sources

https://www.geeksforgeeks.org/introduction-to-kotlin/

https://kotlinlang.org/docs/faq.html#:~:text=Kotlin%20has%20both%20object%2Doriented,doing%20or%20exploring%20functional%20programming

https://www.tekrevol.com/blogs/java-vs-kotlin/

https://www.infoworld.com/article/3224868/what-is-kotlin-the-java-alternative-explained.html