A Study of Kotlin

David Wolfley

April 16 2024

1 History

Kotlin is a programming language created by JetBrains, a company from the Czech Republic founded in August of 2000. The company normally creates IDEs for different programming languages. Most notably in this case is IntelliJ IDEA, which is used for languages that make use of the Java Virtual Machine, including but not limited to Java, Scala, Kotlin, and Groovy. As the name Java came from the name of an island in Indonesia, the name "Kotlin" comes from an island in Russia. Development for the language started in 2010, as the first commit to the Git repository was on November 8, 2010.

The Java Virtual Machine, or JVM, was released by Sun Microsystems (now owned by Oracle) in 1994. The JVM is an interpreter that runs code (originally just Java code) that has been compiled to 'bytecode,' which is an intermediate language.

Oracle, the company that currently owns the rights to Java and the JVM, had an interview with Andrey Breslav, the Lead Language Designer for Kotlin at JetBrains before its release. This interview included information mostly about Kotlin in comparison to other languages that use the JVM. For example, the differences between Kotlin and Scala, and the differences between Kotlin and Java. Some points that were made include: Kotlin has much less features than Scala, while Scala focuses on giving as much power as possible to library designers. Kotlin also has stronger type inference than Java, meaning you do not have to specify the types of specific variables and parameters as much. The overall goals of JetBrains when designing Kotlin were to make a 'general purpose' language, that can be used for all types of programming, which is easier to learn than Java, and also requires less code to perform the same task, while also to encourage more people to buy IntelliJ IDEA by initially only including the language system within installs of either IntelliJ or Android Studio, an IDE for Android development also created by JetBrains.

In 2019, Google declared Kotlin as the favored programming language for Android applications. Because of this, the community for the language grew immensely, and the future development of the language shifted to prioritize Android development over all other types of applications that could be made by Kotlin. Of course, the language can still be used for other purposes. For example, the current version of IntelliJ is written in Kotlin. The command-line compiler for Kotlin can be found here:

https://github.com/JetBrains/kotlin/releases/tag/v1.9.23

However, it is advised to write and run Kotlin code through IntelliJ IDEA, which is JetBrains' IDE for Java. This is because the Kotlin language system is included in the default install for IntelliJ, which can be found here:

https://www.jetbrains.com/idea/

JetBrains also has released a 'getting started' guide, which includes tutorials on the syntax and unique features of the language, which is found here:

https://kotlinlang.org/docs/getting-started.html

Lastly, the official specification for Kotlin is found here:

https://kotlinlang.org/spec/introduction.html

Kotlin is a language that is actively being updated, and JetBrains is still the team in charge of development. Here are some notable versions of the language:

- 1.0 released 2/15/2016: first officially stable release
- 1.2 on 11/28/2017: allowed people to share code between the JVM and JavaScript
- 1.7 on 6/9/2022: released the Kotlin K2 compiler in Alpha, which changes the way that Kotlin code compiles to JVM bytecode, also added type inference operators
- 1.9 on 7/6/2023: K2 compiler now in Beta
- 2.0 (not released yet, but first the pre-release build was released last week, on 4/9/2024): K2 compiler is now stable, and there are major changes to scope, type checking, and null safety.

2 About The Language

In the past, programming languages have had to focus on a single programming style out of necessity, between functional, imperative, object-oriented, and procedural programming. As Kotlin is a newer language, the design philosophy was to make it as versatile as possible. As such, the language has built-in support for all paradigms of programming, which also means that using multiple styles in the same program is possible. It allows for functional programming through the use of lambda functions. It allows for imperative programming through if, when and for statements. It allows for the ability to write programs using classes, which supports object-oriented programming. Lastly, it also allows for procedural programming, as variables and functions can be defined without needing classes.

The official specification for Kotlin does not contain any information about binding, as it is exactly the same as the binding used by the JVM. Therefore, everything about binding can be found in the official specification for the JVM. Because the JVM is an interpreter, it, and by extension every JVM language, handles linking and loading dynamically. This means that it all happens during runtime rather than before.

One major goal of JetBrains when designing the language was to make Kotlin more consise and less bloated than Java, while also allowing for all types of programming unlike Java, which is primarily object-oriented. For an example of how much more consise it is, here is an example of a simple class file in Java:

```
public class MyClass {
private int myField;

    public MyClass(int myField) {
this.myField = myField;
}

    public int getMyField() {
return myField;
}

    public void setMyField(int myField) {
this.myField = myField;
}
}
```

And here is a class file with the exact same properties as defined in Kotlin:

```
class MyClass(private var myField: Int) {

fun getMyField() = myField

fun setMyField(value: Int) { myField = value }
}
```

3 Types

Kotlin's official specification states that the main properties of the type system are as follows:

- Hybrid static, gradual, and flow type checking
- Null safety
- No unsafe implicit conversions
- Unified top and bottom types
- Nominal subtyping with bounded parametric polymorphism and mixed-site variance

The first property to discuss is "Hybrid static, gradual, and flow type checking." On the surface, Kotlin has static types. All variables and values must be assigned a type when initialized, and type checking occurs statically, meaning type errors are caught when compiling rather than during runtime. There are a few exceptions, however. When defining a function, the return type is optional. If a return type is not specified, the language system sees the return type as 'Unit,' which is an optional object of any type, including null. Kotlin is also somewhat adaptive depending on the framework being used. For example, when compiling to JavaScript, variables can be defined with the type 'dynamic,' which essentially creates a variable that is never passed through the type checker, and any function of any class can be called on these objects with no immediate error. This can create problems during runtime, therefore it requires more awareness when using dynamic variables.

When it comes to null safety, Kotlin requires all variables that could possibly be given null values to specify that they could be null. For example, when creating a String object.
var x: String = "Whatever"
var x: String? = "Whatever"
If there is any point throughout the program where that variable could be turned into null, it must be initialized using $\langle T?\rangle$. On the other hand, the '!!' operator converts a value to a non-nullable type, and therefore throws an exception if the value is null. When it comes to type conversion, you can use the 'as?' operator to attempt to convert an object into a different type, and rather than throwing an exception if it fails, it will instead return null.

4 Strengths and Weaknesses

One of the greatest strengths of Kotlin is its interoperability with many different systems. This changes depending on the compiler being used, all of which are officially supported by JetBrains. The most common compiler is Kotlin/JVM, which allows for Kotlin to interact with Java. When using Kotlin/JVM, not only can both Java and Kotlin code be compiled to JVM bytecode in the same project, but Java classes are able to be used in Kotlin, with no changes whatsoever to the source class files. By extension, Kotlin is also compatible with every single Java library. Kotlin/Native allows for Kotlin code to be compiled directly to machine language. This makes it compatible with C code and libraries, which it does with a tool called "cinterop." Even though Kotlin/Native does not use the JVM, various binding times remain the same is if the JVM were involved. Kotlin/WebAssembly and Kotlin/JS allow for the language to be used for web development, and are also usable when compiling either to native machine language code or the JVM. This also means that Kotlin is usable for many different types of programs, such as mobile applications for both ios and android, web applications, and even desktop applications. For example, the current version of IntelliJ Idea was written in Kotlin after the release of the language.

One weakness of Kotlin is its performance. Kotlin code takes longer to compile than Java, which grows exponentially with the complexity of the program. It compiles to the same exact bytecode that a Java program with the same functionality would, but the difference in compile time is possibly due to the null safety and more consise code, which requires more work from the compiler. However, many of the other weaknesses of the language are boiled down to the fact that it is new. It's age creates a some problems when using and learning the language. Firstly, the community is less developed, resulting in less assistance with major issues. This both makes it harder to use Kotlin for programs that people haven't used it for in the past, and also means that there are less third party libraries for the language. This is circumvented by the use of third party Java libraries, however using those libraries can lead to strange errors that are near impossible to diagnose. The language being newer also means it is generally less stable. Even though the 2.0 version is being released sometime soon, the language actively evolving means that programs written in older versions of the language can eventually become obsolete. Also, the fact that Kotlin is now primarily supported for use in Android development means that future updates for the language could just support this use of the language and can possibly ignore other ways that the language can be used, like for web development or desktop applications.

5 Learning Experience

The best part about learning Kotlin for me has been its interoperability with Java. Java is the language I am the most comfortable with using at the moment, which was a major factor in my ability to succeed with this program. Since the fact that I could use Java was the only thing I was aware of going into this project, it allowed me to be aware of the types that exist in Kotlin, as many of them are the same as those in Java. When it comes to the unique features of Kotlin, the ones that stand out to me are null safety and the support for functional programming. As I see it, Kotlin's null safety allows for a majority of errors that occur in Java applications to not occur, as every situation where null could possibly appear as a value is already handled. This creates situations similar to what we had when learning ML, where not getting an error likely results in the program working the way it is intended.

I did, however, have major problems when working on this application. These spawned not from the Kotlin Standard Library, but instead the primary UI library for Kotlin, which is Jetpack Compose. This library is primarily used for Android applications, but can also be used for desktop application, which is what I used it for. It created tough situations where I couldn't avoid duplicate code because of issues with scope. In these situations, I was required to create individual ui elements inside of the scope of the 'column' of the screen they were in, which required me to use duplicate code inside every button in the program. My knowledge of Java also became a problem, as I would often resort to Java syntax, and it would sometimes take a long time to realize what I did wrong.

Overall, I believe the positives of the language outweigh the negatives. Kotlin is a very interesting, very versatile language to use, and I would absolutely consider using it again if given the chance.

6 References

Krill, P. (2011, July 22). *JetBrains Readies JVM-based language.* InfoWorld.
https://www.infoworld.com/article/2622405/jetbrains-readies-jvm-based-language.html

Kotlin programming language. *Why Teach Kotlin.* (n.d.).
https://kotlinlang.org/education/why-teach-kotlin.html

Akhin, M., Belyaev, M. (n.d.). *Kotlin Language specification.*
https://kotlinlang.org/spec/introduction.html

Java Virtual Machine Specification. Oracle. (n.d.).
https://docs.oracle.com/javase/specs/jvms/se8/html/index.html