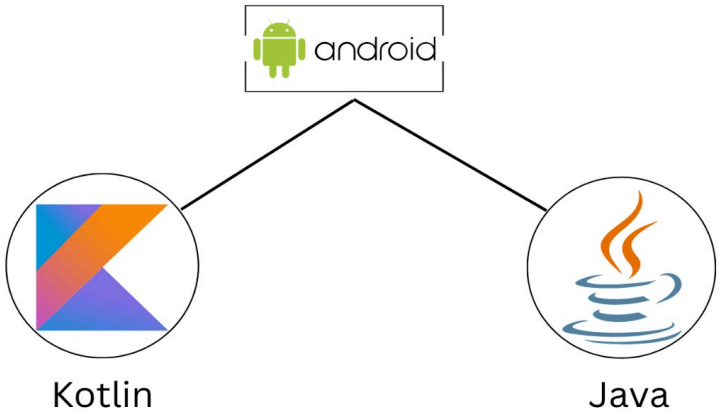
Abstract: This paper discusses the comparison and contrast between Java and Kotlin programming languages for creating Android applications. Java has been the dominant language used for Android app development, but Kotlin has emerged as a viable alternative. Various aspects of both languages, including syntax, null safety, type inference, and functional programming has been analyzed. It will assist developers to decide which language to use for their projects by providing a comprehensive analysis of the strengths and weaknesses of both languages. The findings of this study suggest that Kotlin has many advantages over Java, such as its concise syntax, null safety features, and support for functional programming. As such, developers may consider using Kotlin as an alternative to Java for Android app development.

Introduction: Smartphone applications are a fast-expanding subset of the worldwide mobile market in the rapidly developing world of technology. Mobile apps are developing quickly to provide users with a rich and quick user experience. This paper has covered the use of Java as the programming language for Android applications. When comparing Java with Kotlin, it can be seen that Kotlin is a new and exciting area of Android technology that has the potential to surpass Java.



Android: Google developed Android as an open-source operating system for mobile phones, which runs on the Linux operating system. The system comprises the operating system, middleware, and application and user interface software [12]. Android is highly popular among cell phone users and was originally designed for touch screen mobile devices, including tablets and smartphones.

In addition to its general operating system, Google has also created custom interfaces for various devices. For example, Android TV is tailored for televisions, Android Auto is optimized for cars, and Wear OS is designed for wristwatches [14]. Additionally, other electronic devices, such as game consoles, digital cameras, and PCs, also use versions of the Android OS. There are numerous versions of Android. Every Android version has a dessert-inspired name. Cupcake is the name of the initial named release of Android [16]. Other Android versions include Marshmallow, Nougat, Oreo, Lollipop, Kitkat, and Jellybeans. Although Android Nougat is more widely used, Android Oreo, released in February 2018, is the most newest version of Android, according to [15].

Few users are aware of specific security flaws that are present in Android. Before 2016, we had access to thousands of Android applications that anyone could upload to the Android Market

without having to go through rigorous security tests. Because of this, Android is a popular target for cybercriminals. To address the aforementioned problem, Google has started the Play Protect Project [13].

Java: The programming language Java was initially introduced in 1995 by Sun Microsystems. It is widely used across various devices, including smartphones and mainframe computers.

It works on desktop computers and even Raspberry Pi. Instead of relying on a "virtual machine," which can interpret Java bytecode, Java doesn't produce native processor code during compilation. [1][23]. There must be a virtual machine (VM) incorporation for each console that supports Java.

Android's main virtual machine is called Dalvik, but Google has also released a preview of their next-generation ART virtual machine. These virtual machines are responsible for translating the program's bytecode, which is similar to the machine code used by CPUs and consists of a series of procedures. VMs use several technologies, including just-in-time (JIT) and ahead-of-time (AOT) compilation, to speed up program execution.

Android apps can be deployed on platforms like Windows, Linux, and OS X by compiling the source code into bytecode using the Java compiler. The resulting bytecode runs on the Android virtual machine. In contrast, iOS uses a native compiler to convert Objective-C into ARM machine code. [2]

[17] states that Java is practised for Android development since it is popular among developers and does not have the difficulties of pointer arithmetic. Because the code successfully complied on VMs, it does not need to be recompiled for each device it is used on. Although Java's speed can be a drawback, its popularity and advantages much surpass it.

Kotlin: The Java Virtual Machine supports the statically-typed programming language Kotlin (Android and Kotlin), which may also be turned into JavaScript source code. In February 2016, it was made accessible to the general public. The Kotlin Island, which is close to St. Petersburg, inspired the name of the software, which was primarily developed by a group of JetBrains programmers encamped in Saint Petersburg, Russia. In actuality, Kotlin Island in Saint Petersburg is where the name originated[18].

The Android team at Google announced in May 2018 that Kotlin is now the official language for Android development.

While developers have used Kotlin to create Android apps previously, Google has recently declared full support for the language.

Java and Kotlin are both viable programming languages for creating Android apps. However, the reason for the recent shift towards Kotlin remains a significant question.

Literature: Panchaland, R.K., and Patel, A.K., conducted a comparative study of Kotlin and Java for Android development, citing previous research in the field. Their work suggests that Kotlin could be a promising language for creating mobile apps [18].

Android Authority released a video on June 26, 2017, entitled "10 Reasons to Try Kotlin for Android Development," which highlights the unique features that Kotlin offers for building Android applications. These features include extensions and null pointer handling [24].

Holla, S., and Katti, M.M., investigated security challenges and potential future developments for Android applications in their research [25]. Among the issues they discussed were the risks of downloading virus-infected apps from the Android Market and the potential use of additional sensors in future devices to enhance Android security.

Objective: This paper compares and contrasts Java with Kotlin for Android applications with a conclusion. We have contrasted Java and Android with several fields. This paper makes an effort to examine the several aspects of both Java and Kotlin before deciding which programming language is ideal for developers.

Research Methodology: This study is an explanation that clarifies the use of Java in Android development and highlights Kotlin as a java substitute in Android applications. The secondary data for this qualitative study was acquired from various sources that were released in electronic media.

Comparison between Java and Kotlin

Extension Functions: In numerous programming languages, a new contemporary class is extract if more functionality in an existing class are needed. A function which is member to a class and has definition out the front of the class is called an extension function. The example provided in [19] can be used to further explain extension function.

As in the example, a function is required. The initial and last characters of a string must be removed before the String class can return a new string; this function is not included in the String class. The functionality of the specified class is created by the extension function, which is declared outside of the class and extends the predefined functionalities. The following can be added to the function:

Example:

```
fun String.removeFirstLastChar(): String = this.substring(1, this.length - 1)
fun main(args: Array<String>) {
    val mString= "Hello Android"
    val res = mString.removeFirstLastChar()
    println("First character is: $res")
}
```

Kotlin offers the option to add additional functionality to a class without needing to inherit from it or make use of any design patterns, such as Decorator. Extensions are specialised declarations that are used for this. Extension properties and extension functions are supported by Kotlin. [4]

Java does not contain this Extension function. Android frameworks are frequently used to make extension functions functional.

But occasionally the Android framework makes things challenging. The only option offered by Java is to build wrappers. [5]

However, Kotlin contains the benefit of an extension function, that ultimately remove the challenges presented by the Android architecture.

Checked Exceptions: Try...catch blocks are used by Java to handle runtime exceptions. Mostly checked exceptions are used.

An exception of this type must be handled, or at least proclaimed, in the procedure in which it is thrown.

The try and catch block in JAVA has the syntax listed below [20].

```
try
{ // some code }
catch (e: SomeException)
{ // handler }
finally
{ // optional finally block }
```

A method in Java can have zero or more catch blocks and one or zero finally blocks. It is compulsory to handle an exception using either a catch block or a finally block. If a checked exception is thrown by a method's code in Java, the method must either handle the exception or declare it using the throws keyword.

There is no checked exception in Kotlin. Kotlin's exception classes are all descended from the Throwable class. Every exception has an optional cause, message, and stack trace.[6]

Kotlin supports handling exceptions using throw expression [20].

```
fun f1(message: String): Nothing
{ throw IllegalArgumentException(message) }
val pname = person.name ?: throw IllegalArgumentException("Name required")
```

Whether a checked exception is preferable to an unchecked exception is up for dispute. Although some Kotlin applications manage to incorporate checked exceptions, here we list the benefits of not doing so.

The logic or flow of the code may be broken by checked exceptions [21]. Using checked exceptions might result in lost code flow, especially in programmes with several callback functions.

Second, checked exceptions in big software systems result in worse productivity and little to no improvement in the quality of the code [20].

Constructors: A secondary function Object() { [native code] } and a primary function Object() { [native code] } are both possible in Kotlin [8]. To declare secondary constructor, use the "constructor" keyword. It often pertains to the primary constructor [7].

Using a secondary constructor as an example:

```
class Student {
    val name: String
    val roll: Int
    val marks: Int
    private var elective = false
    constructor(name: String, roll: Int, marks: Int) {
        this.name = name
        this.roll = roll
        this.marks = marks
    }
    constructor(name: String, roll: Int, marks: Int, elective: Boolean)
    : this(name, roll,marks) {
        this.elective = elective
    }
}
```

The primary constructor in the example above has three parameters, while the secondary constructor has four.

Java constructor overloading would have resulted in the following code:

```
class Student {
    String name;
    Int roll;
    Int marks;
    Int elective;
    Student (String name, Int roll, Int marks) {
        this.name = name;
        this.roll = roll;
        this.marks = marks;
        this.elective = 0;
    }
    Student (String name, Int roll, Int marks, Int elective) {
        this.name = name;
        this.roll = roll;
        this.marks = marks;
        this.elective = elective;
    }
}
```

Java doesn't have this secondary constructor feature. The secondary constructor's benefit is that it cuts down on the number of lines of code.

Null Safety: Retrieving a member of a null reference will cause a null reference exception, which is one of the most frequent problems in numerous programming languages, including Java. This is comparable to a NullPointerException, in Java [3][9].

To deal with the NULL pointer scenario, Kotlin makes advantage of a feature called Null Safety. Kotlin does not raise a NullPointerException unless it is expressly requested [9].

The Java code is below.

```
public static void main(String []args){
    String name= null;
    System.out.println(name.length());
}
```

Output:

```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke
    "String.length()" because "<local1>" is null
```

Here is some Kotlin code.

```
fun main() {
    var name: String? = null
    println(name?.length)

}
```

Output:

```
null
```

In contrast to Java, Kotlin does not disrupt the flow of the code when the Null Pointer error occurs. The output is displayed as NULL.

Lazy Loading: Computer programmes employ lazy loading to delay initialising an object until a time when it is required. As a result, the lazy-loading feature speeds up loading.

Unlike Java, Kotlin has the charactertics of lazy loading. Java does not support lazy loading, therefore a lot of unnecessary stuff is loaded when the application first launches, slowing down the loading time.

Conclusion: We concluded from our analysis that Java and Kotlin each have advantages and drawbacks of their own.

For novices, Java is a superior choice because of the following factors:

Developers use the language Java a lot because it is so well-liked. The amount of work taking place globally is vast, and Android development is but a drop in the bucket. Knowing JAVA is therefore more useful for a novice than Kotlin because it broadens the range of contingency.

Second, there is a substantial commanwealth of Java programmers, so when we get stuck, we can always discover solutions. This is crucial since, as beginners, we frequently run into technical issues and may not know where to turn when we get stuck. When we look for solutions to Java-related issues, we are guaranteed to find them; this cannot be stated for Kotlin, a still-emerging programming language.

The same cannot be true for Kotlin, whereas there are more free and paid tutorials, books, and courses available to educate us Android development using Java.

From a developer's perspective, Kotlin would be favoured for the reasons listed below [10]:

1. **Language and environment are mature:**
   Unlike other programming languages, the Kotlin release went through several stages before the official 1.0 version was released.
   This may imply that during the evolution to the final level, all the likely problems frequently encountered in other programming languages have already been addressed. [5]

2. **Makes Android Development easier:**

Programming is made simpler and Android apps are improved using Kotlin. A contemporary programming language is Kotlin. For those who create Android apps, it opens up a wide range of opportunities, increasing productivity. [5]

3. **Kotlin aids in minimizing flaws and errors in the code:**
   The Kotlin compiler strives to fail quickly whenever feasible, which makes it much easier to find flaws. The Kotlin compiler does a number of checks to prevent runtime mistakes and lower the cost and work needed to fix issues. [11]

4. **Kotlin code is safer:**
   By adopting Kotlin, it is simple to avoid common programming errors by design, which leads to fewer system errors and application crashes. This demonstrates how Kotlin code is fundamentally safer than any other kind of computer code. [11]

5. **Kotlin is more compact:**
   In many situations, Kotlin is far more concise than any other programming languages; it enables us to tackle the same problems with less code. Engineers can develop, read, and alter code more effectively and efficiently as a result of improved code maintainability and readability. [11]
   A developer is free to change at any time because Android provides the support to convert a project to Kotlin.
   Kotlin is desirable because it boosts output. In Kotlin, a class that requires 50 lines of Java code can be created in just a handful. It is possible to avoid boilerplate code, such as the necessity to define the setters(), equals(), hashCode(), or toString() methods. Kotlin is capable of producing each of these.
   The number of Kotlin lines of code required to produce the same outcome is significantly lower than the number of Java lines of code. Below is an illustration of this, which is taken from [22]:

   Java

```java
class User {
    private final String firstName;
    private final String lastName;
    private final int Age;

    public User(String firstName, String lastName, int Age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.Age = Age;
    }

    public String getFirstName() { return firstName; }

    public String getLastName() { return lastName; }

    public int getAge() { return Age; }

    public String toString() { return firstName + " " + lastName + ", age " + Age; }
}

public class Main {

    public static void main(String[] args) {
        System.out.println(new User( firstName: "Akshit",  lastName: "Sharma",  Age: 20));
    }
}
```

Kotlin

```
public class User(val firstName:String, val lastName:String, val age:Int){
    fun converttoString()="$firstName $lastName ,age $age"
}

fun main(args:Array<String>){
    println(User( firstName: "Akshit", lastName: "Sharma", age: 20).toString())
}
```

References:

[1] E. Obugyei, 2016, “Kotlin for Android: An Introduction”, viewed on 25th April, 2018 from https://www.raywenderlich.com/132381/kotlin-for-androidan-introduction.

[2] C. Singh, n.d., “Introduction to Java Programming”, viewed on 1st May, 2018 from https://beginnersbook.com/2013/05/java-introduction/

[3] A. Sinicki, 2018, “An Introduction to Kotlin for Android development”, viewed on 25th April, 2018 from https://wwwhttps://www.androidauthority.com/introductionto-kotlin-for-android-775678/

[4] “Kotlin”, viewed on 1st May, 2018 from https://kotlinlang.org/docs/reference/extensions.html

[5] A. Leiva, n.d. , “Extension functions in Kotlin: Extend the Android Framework (KAD 08)”, viewed on 1st May, 2018 from https://antonioleiva.com/extension-functions-kotlin/

[6] ChikeMgbemena, n.d., viewed on 7th May, 2018 from https://code.tutsplus.com/tutorials/kotlin-from-scratchexception-handling--cms-2

[7] “Kotlin”, viewed on 8th May 2018, from https://kotlinlang.org/docs/reference/classes.html

[8] D. Odalodic, June, 2017, “Dusan Odalodic @ JVM”, viewed on 8th May 2018, from https://odalinjo.wordpress.com/2017/06/25/primary-andsecondary-constructors-in-kotlin/

[9] M. Daga, May, 2018, “Java vs Kotlin: Which Programming Language Is Better for Android Developers?”, viewed on 8th May , 2018 from https://dzone.com/articles/java-vs-kotlinwhich-programming-language-is-better

[10] “What are the advantages of Kotlin over Java? “, viewed on 8th May 2018, from “https://www.quora.com/in/What-arethe-advantages-of-Kotlin-over-Java

[11] P. Sommerhoff, January, 2018, “Kotlin vs. Java: 9 Benefits of Kotlin for Your Business”, viewed on 7 May, 2018, from https://business.udemy.com/blog/kotlin-vs-java-9-benefitsof-kotlin-for-your-business/

[12] G. Suite, n.d., “Android”, viewed on 9 May, 2018 from https://www.engineersgarage.com/articles/what-is-androidintroduction

[13] I. Majocha, December, 2017, “Report: Top Android Security Problems in 2017”, viewed on 9 May, 2018 from https://dzone.com/articles/report-top-android-securityproblems-in-2017

[14] Developers, n.d., “About the platform”, viewed on 9 May, 2018 from https://developer.android.com/about/

[15] Fossbytes, n.d., “Most Popular Android Versions In February 2018 (Always Updated List)”, viewed on 9 May, 2018 from https://fossbytes.com/most-popular-androidversions-always-updated/

[16] Turbofuture, April, 2016, “Android Version Names: Every OS from Cupcake to Marshmallow”, viewed on 9 May, 2018 from https://turbofuture.com/cell-phones/CupcakeDonut-Eclair-Froyo-Gingerbread-Honeycomb-Android-OSVersion-Codenames-and-Why

[17] “Why does Android use Java?”, viewed on 9 May, 2018 from https://stackoverflow.com/questions/3560963/whydoes-android-use-java

[18] R.K. Panchal, and, A.K. Patel, 2017, A comparative study: Java Vs kotlin Programming in Android , in International Journal of Innovative Trends in Engineering & Research, September 2017, vol 2 Issue 9, pp 4 – 10.

[19] Programiz, n.d., “Kotlin Extension Function”, viewed on 9th May 2018, from https://www.programiz.com/kotlinprogramming/extension-functions

[20] “Kotlin”, viewed on 8 May, 2018 from https://kotlinlang.org/docs/reference/exceptions.html

[21] E. Petrenko, n.d., “Catching exceptions with less code in Kotlin”, viewed on 9 May, 2018 http://jonnyzzz.com/blog/2017/02/15/catchall/

[22] Hype.codes, 2017, “ Kotlin Vs Java”, viewed on 9 May, 2018 from https://hype.codes/kotlin-vs-java

[23] Schildt, The Complete Reference Java, Seventh Edition, Chapter 1, page 9 May, 2018

[24] Android Authority, 2017, “10 reasons to try Kotlin for Android development”, viewed on 9th May, 2018 from https://www.youtube.com/watch?v=LEi1ecigDFE

[25] S. Holla and M.M. Katti, 2012, ANDROID BASED MOBILE APPLICATION DEVELOPMENT and its SECURITY, in International Journal of Computer Trends and Technology, 2012, Vol 3, Issue 3, pp 486-490.