Comparing Three programming Languages

Sarthak Kothari
Department of Computer Science,
Lakehead University
Thunder Bay, Ontario
skothar1@lakeheadu.ca

***Abstract*—There is often a big discussion on which language excels the other, but we can never reach a conclusion due to lack of comparing sources and definite research work. In this paper we will try to reach a definite or good enough answer to what will be more efficient language than the other based on many factors. Our investigation will try to investigate several aspects of these programming languages and try to reach a conclusion using these factors like program length, programming effort, runtime efficiency, memory consumption, reliability and main advantages and disadvantages of some standard programming languages.**

Keywords—Null safe, memory utilization, inline function, interoperability, JVM based programming languages, Kotlin, Groovy, Scala.

I. Introduction

When we talk about Android app development, the first programming language that comes to our mind is 'Java'. Though most Android apps are written in Java, developers should know that it is not the only language to write code. Android app can be written in any language that can be compiled and run on Java Virtual Machine (JVM).

In this paper I will try to compare 3 JVM based programming languages based on many factors and try to find which language is better for us even though it is difficult to compare languages as each of them have there own good and bad which changes depending on the situation. So, lets try to find the answer using basic comparison.

II. Coding Languages

A. SCALA Language

Scala is a general-purpose programming language providing support for functional programming and a strong static type system. Designed to be concise, many of Scala's design decisions aimed to address criticisms of Java.

Scala source code is intended to be compiled to Java bytecode, so that the resulting executable code runs on a Java virtual machine. Scala provides language interoperability with Java, so that libraries written in either language may be referenced directly in Scala or Java code. Like Java, Scala is object-oriented, and uses a curly-brace syntax reminiscent of the C programming language. Unlike Java, Scala has many features of functional programming languages like Scheme, Standard ML and Haskell, including currying, type inference, immutability, lazy evaluation, and pattern matching. It also has an advanced type system supporting algebraic data types, covariance and contravariance, higher-order types (but not higher-rank types), and anonymous types. Other features of Scala not present in Java include operator overloading, optional parameters, named parameters, and raw strings. Conversely, a feature of Java not in Scala is checked exceptions, which has proved controversial.

The name Scala is a portmanteau of scalable and language, signifying that it is designed to grow with the demands of its users.

B. KOTLIN Language

This Kotlin is a cross-platform, statically typed, general-purpose programming language with type inference. Kotlin is designed to interoperate fully with Java, and the JVM version of its standard library depends on the Java Class Library, but type inference allows its syntax to be more concise. Kotlin mainly targets the JVM, but also compiles to JavaScript or native code (via LLVM). Kotlin is sponsored by JetBrains and Google through the Kotlin Foundation.

Kotlin is officially supported by Google for mobile development on Android. Since the release of Android Studio 3.0 in October 2017, Kotlin is included as an alternative to the standard Java compiler. The Android Kotlin compiler lets the user choose between targeting Java 6 or Java 8 compatible bytecode.

Kotlin has been Google's preferred language for Android app development since 7 May 2019. Google announced it as the official language for Android Development on 2017 Google IO.

C. *Groovy Language*

Apache Groovy is a Java-syntax-compatible object-oriented programming language for the Java platform. It is both a static and dynamic language with features similar to those of Python, Ruby, and Smalltalk. It can be used as both a programming language and a scripting language for the Java Platform, is compiled to Java virtual machine (JVM) bytecode, and interoperates seamlessly with other Java code and libraries. Groovy uses a curly-bracket syntax similar to Java's. Groovy supports closures, multiline strings, and expressions embedded in strings. Much of Groovy's power lies in its AST transformations, triggered through annotations.

Groovy 1.0 was released on January 2, 2007, and Groovy 2.0 in July, 2012. Since version 2, Groovy can be compiled statically, offering type inference and performance near that of Java. Groovy 2.4 was the last major release under Pivotal Software's sponsorship which ended in March 2015. Groovy 2.5.8 is the latest stable version of Groovy. Groovy has since changed its governance structure to a Project Management Committee in the Apache Software Foundation

simulations and many more applications.

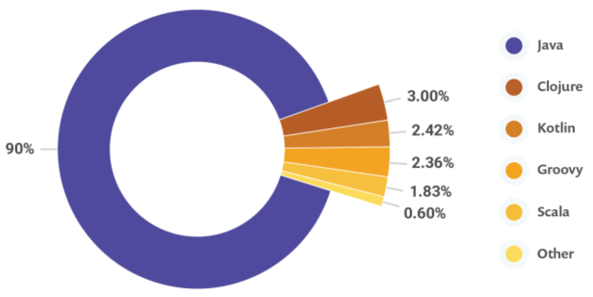


Fig 1 Top choice for JVM developers.

III. COMPARISIONS

We will try to differentiate between these three languages based on many features and will try to provide a suitable conclusion which can be easily derived.

A. *Market Trend:*

The image will show the popularity of the JVM programming languages in the IT industry in the recent year.

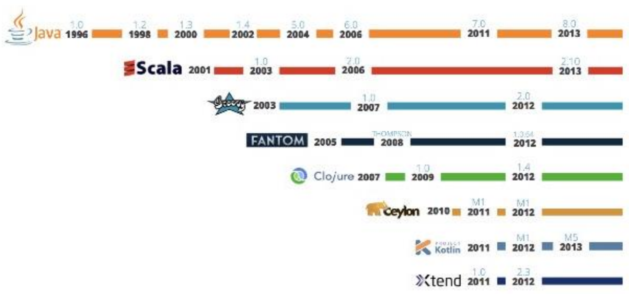


Fig 2. History of JVM programming languages.

From the above figure we can see that there has been constant improvement and growth in JVM programming languages. Kotlin being the most recent programming language gaining the lead compared to Scala and Groovy.

B. *Origin*

Scala and Kotlin were developed by Jetbrains whereas Groovy was developed by Apache.

C. *Non-object Primitives*

Java & Groovy does have primitives which are not really objects. There are separate wrapper object classes available for the each primitive type. On the contrast, Scala & Kotlin does not have primitives at all. All number types are objects. In other words, everything is object in Scala & Kotlin.

D. *Type Inference/Optionally Typed*

Type inference means automatic detection of data type instead of statically typed data type.

```
Groovy:
        Def s=”It’s a Binary!”
        println s

Scala:
        Def s=”It’s a Binary!”
        println(s)

Kotlin:
        val s=”It’s a Binary!”
        println(s)
```

E. *Duck Typing*

Duck typing is way to determine behavior of class on basis of method presence instead of type. So type does not determine at compile time if certain behavior is suitable or not. At runtime, method presence is checked in object & executed or else error. Duck typing is only supported in Groovy.

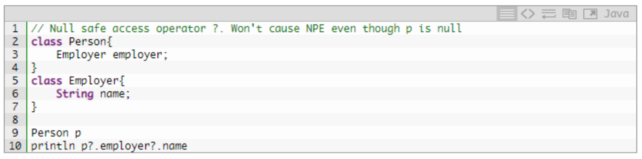
Others i.e. Java, Kotlin, Scala doesn’t support duck typing.

F. *Null Safety*

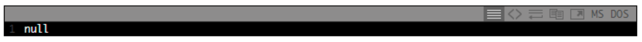
Groovy & Kotlin does provide few ways to gracefully prevent NullPointerException as shown in below exception. Java & Scala do not have features to specifically prevent “Null Pointer Exception” & developers are expected to handle it explicitly.

1] Groovy:

Null Safe access operator ? which is quivalent to if(p not null) then p.employer null



```
// Null safe access operator ?. Won't cause NPE even though p is null
class Person{
    Employer employer;
}
class Employer{
    String name;
}

Person p
println p?.employer?.name
```

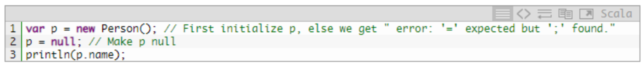


```
null
```

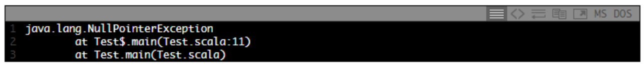
Fig 4.a Groovy.

2] Scala:

No null safety. Standard null checks needed like Java.



```
var p = new Person(); // First initialize p, else we get " error: '=' expected but ';' found."
p = null; // Make p null
println(p.name);
```

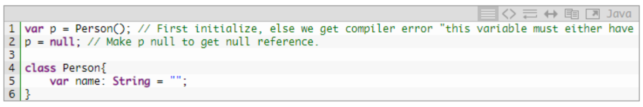


```
java.lang.NullPointerException
        at Test$.main(Test.scala:11)
        at Test.main(Test.scala)
```

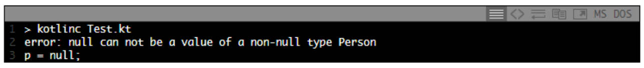
Fig 4.b Groovy.

3] Kotlin:

Compile time check for null assignment.



```
var p = Person(); // First initialize, else we get compiler error "this variable must either have
p = null; // Make p null to get null reference.

class Person{
    var name: String = "";
}
```



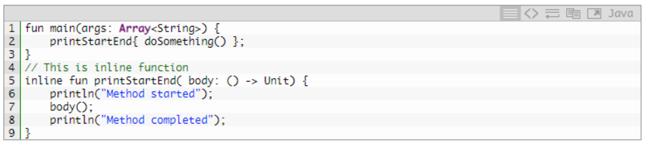
```
> kotlinc Test.kt
error: null can not be a value of a non-null type Person
p = null;
```

Fig 4.b Groovy.

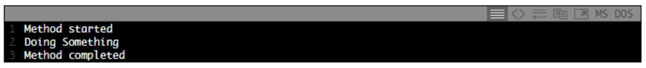
G. *Inline Functions*

Inline function instructs compiler to insert complete body code of the function wherever that function is called in code. This is a lightweight alternative to higher order functions which calls for creation of object wherever they are used. With inline function, code is placed at call site at compile time. This is generally used for smaller behaviors.

Kotlin supports inline functions. Couldn’t see any such support in Groovy, Scala & Java.



```
fun main(args: Array<String>) {
    printStartEnd{ doSomething() };
}
// This is inline function
inline fun printStartEnd( body: () -> Unit) {
    println("Method started");
    body();
    println("Method completed");
}
```



```
Method started
Doing Something
Method completed
```

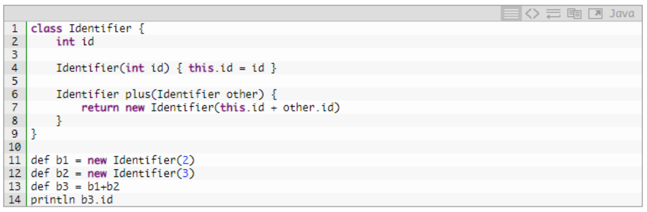
Fig 5. Kotlin Example.

H. *Operator Overloading*

Operator overloading is a way to add additional purpose for operators depending on its arguments.

1) **Groovy**

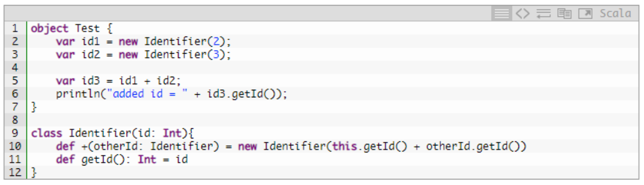
Groovy has special methods like plus() which can be overridden to implement operator behavior for given class’ objects.



```
class Identifier {
    int id

    Identifier(int id) { this.id = id }

    Identifier plus(Identifier other) {
        return new Identifier(this.id + other.id)
    }
}

def b1 = new Identifier(2)
def b2 = new Identifier(3)
def b3 = b1+b2
println b3.id
```

2) **Scala**

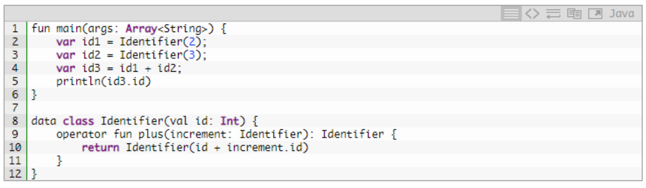
In Scala operators are not really operators but they are actually methods unlike Java. So basically operator overloading is nothing but defining method/function with name as operator like +.



```
object Test {
    var id1 = new Identifier(2);
    var id2 = new Identifier(3);

    var id3 = id1 + id2;
    println("added id = " + id3.getId());
}

class Identifier(id: Int){
    def +(otherId: Identifier) = new Identifier(this.getId() + otherId.getId())
    def getId(): Int = id
}
```

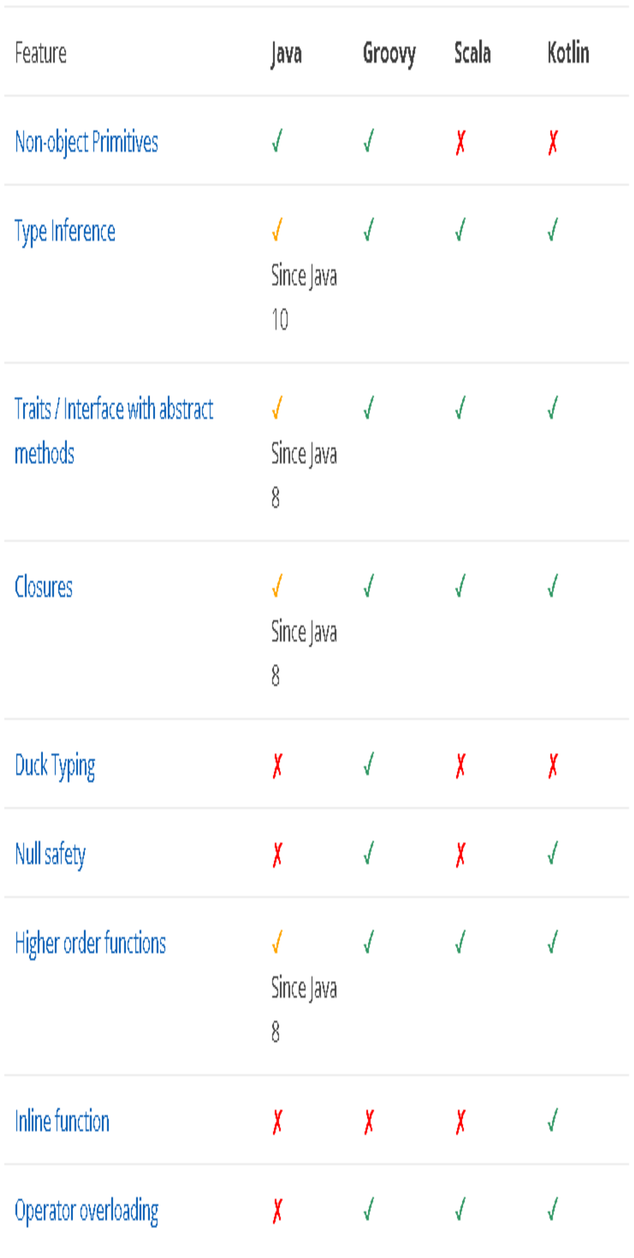
3) **Kotlin**

Kotlin compiler translates + operator into plus() method & then execution happens like any other method.



```
fun main(args: Array<String>) {
    var id1 = Identifier(2);
    var id2 = Identifier(3);
    var id3 = id1 + id2;
    println(id3.id)
}

data class Identifier(val id: Int) {
    operator fun plus(increment: Identifier): Identifier {
        return Identifier(id + increment.id)
    }
}
```

This is a quick table of summary of comparison. Scroll down further for in depth examples & actual code comparison.



Feature	Java	Groovy	Scala	Kotlin
Non-object Primitives	✓	✓	✗	✗
Type Inference	✓ Since Java 10	✓	✓	✓
Traits / Interface with abstract methods	✓ Since Java 8	✓	✓	✓
Closures	✓ Since Java 8	✓	✓	✓
Duck Typing	✗	✓	✗	✗
Null safety	✗	✓	✗	✓
Higher order functions	✓ Since Java 8	✓	✓	✓
Inline function	✗	✗	✗	✓
Operator overloading	✗	✓	✓	✓

Fig 6. Features Summary

Above figure gives a succinct idea of the difference of the features of all 3 JVM programming languages : SCALA, KOTLIN and GROOVY. This table help in understanding an overview of the major difference in features for all 3 languages as compared to JAVA language.

IV. Worldwide Intrest of JVM programming language.

Continent Spread:

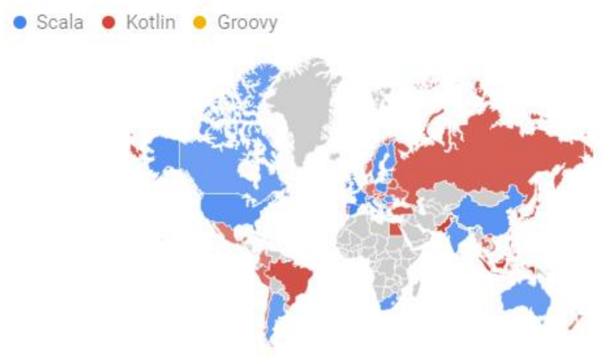


Fig 7.Intrest of JVM programming languages

SCALA Language

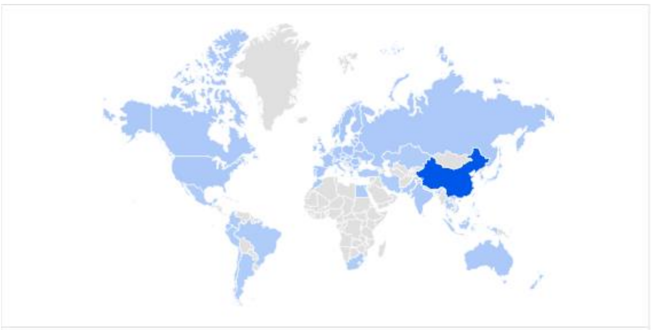


Fig 8. Regionwise intrest for SCALA language

KOTLIN

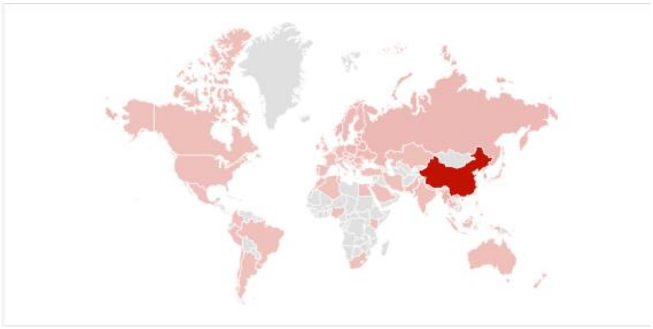


Fig 9. Regionwise intrest for KOTLIN language

GROOVY

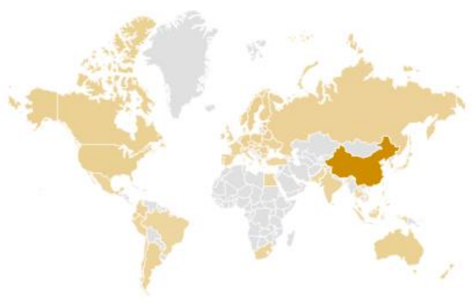


Fig 10. Regionwise intrest for GROOVY language

.

V. CONCLUSION

As per my observation I feel that Kotlin is better then SCALA and GROOVY programming languages. This is because :

1. Kotlin has a practical mix of features from java, C# and other new languages.
2. Kotlin is continually maintained by a commercial company, Jetbrains and it's an official target for building Android applications AND it has an excellent IDE. In a matter of 1–2 years, this will spawn a good developer ecosystem/community that will have a lot of projects coming up in all areas of development.
3. A build tool like Gradle hasn't become popular but I'm expecting a new one or one of the existing ones to become good enough for most projects

REFERENCES

[1] https://en.wikipedia.org/wiki/Scala_(programming_language)

[2] http://www.nicolas-hahn.com/python/go/rust/programming/2019/07/01/program-in-python-go-rust/

[3] https://stackshare.io/stackups/go-vs-python-vs-rust

[4] https://kotlinlang.org/docs/reference/comparison-to-java.html

[5] http://itsallbinary.com/java-vs-groovy-scala-kotlin-code-comparison-of-jvm-languages/#noobjprimitive

[6] https://www.bacancytechnology.com/blog/kotlin-vs-java-which-is-the-best

[7] https://medium.com/@promatics/is-kotlin-better-than-java-is-it-time-to-switch-over-for-android-developers-68d283f0b169

[8] https://en.wikipedia.org/wiki/Apache_Groovy

[9] https://www.whoishostingthis.com/compare/java/jvm-programming/

[10] https://www.quora.com/Which-one-is-better-kotlin-vs-groovy