ISSN (Online) 2456 -1304

International Journal of Science, Engineering and Management (IJSEM)
Vol 2, Issue 12, December 2017

KOTLIN - A New Programming Language for the Modern Needs

[1] Mrs.J.ArockiaJeyanthi, [2] Mrs.T.Kamaleswari
[1] Assistant Professor, [2] Assistant Professor,
[1] Department of Computer Science, A.P.C. Mahalaxmi College for Women, Thoothukudi.

***Abstract:--* Java programming language is a widely used language for the development of Android applications. However recent research proves that this language suffers from certain drawback which is the main reason for the crashes of Android applications. This has paved the way for other alternative languages like Groovy, Scala, Kotlin, etc.. All the other languages mentioned above have their own demerits like Groovy suffers from un safety whereas Scala generates steep learning curve. But Kotlin can be used widely instead of Java almost everywhere and its usage can be widely seen in Android applications development, Server-side development and much more. Our research work analyses how Kotlin can be integrated with the existing Java language. The experimental results prove that this new programming language can reduce the compilation time, execution time and can increase conciseness when integrated with Java.**

***Keywords:--* Programming language, Android applications, Kotlin**

1. INTRODUCTION

The goal of this paper is to give an overview of the Kotlin language and it's wide support in developing Android applications,Server side applications and much more.The paper is also meant to address the significant portions where Java lacked and why Kotlin is going to take over Java real soon.It also discusses about Conciseness ,Interoperability and productivity which are the major things kept in mind when working on any Android project.To get a proper overview of Kotlin let us start the discussion with the features where Java lacks.

II. DISADVANTAGES OF JAVA

Java – an unsafe language
Null Pointer Exception, also referred to as **"THE BILLION DOLLAR MISTAKE"** is one of the well-known drawback of Java.Every other day another security loop hole peeps up in Java that everybody struggles with.Everyone wants clean and simple code, which is self-understandable and easily read by others also, but Java makes it cumbersome to organize code in a concise way. One of the biggest drawbacks of Java is that writing huge sections of code even for the smallest of tasks,. And at last, it turns out to be a pain for the coders with innumerous lines of codes .

The main reason how Kotlin stands out in the ocean of programming languages is that it improves over Java's limitations and positively affects day to day development workflow. It is incredibly powerful and has a handful of things which would attract any coder.

Interoperability
Interoperability is the most outstanding feature in Kotlin's magic box one can call a code written in Java to Kotlin and vice-versa. There is no need to convert the entire project into Kotlin from the very beginning.

Freedom from Null Pointer Exceptions
We need not be afraid of Null Pointer Exceptions, because Kotlin takes NULL value checks from runtime to compile time which means that Null safety is a part of the system itself. All variables are non-null in Kotlin,.

Lesser verbosity
Kotlin helps us to write concise and crisp code to help save ample time and decrease the boilerplate & clustering. The programmer to gest rid of the worrying task of adding semicolons after every statement. This helps in increasing productivity code as well as saves time.

Smart Casts
The coder need not worry about explicitly casting operators because Kotlin's compiler inserts casts automatically wherever needed.

Destructuring Declarations
Multiple variables for an object are declared at a stretch.

Awesome IDE and Plugin support
To get Kotlin install a simpleplugin in Android Studio or even in Eclipse .
To convert our existing Java code to Kotlin we have an amazing plugin "Convert Java file to Kotlin"

III. IS KOTLIN PERFECT

Though Kotlin seem to be the most promising one it also have some flaws.

Sluggish compilation
The compilation speed which is fairly less than its competitive languages is a minor drawback of Kotlin .

Small developer community
For the time being Kotlin still has a small developer community despite its rapid adoption among coders.

Larger package size
The package size of Kotlin is bigger in size as compared to Java.

Java to Kotlin
Around 80% of the Java code converts to Kotlin might be a boon seamlessly. The remaining 20% of the code gets thoroughly scrambled and can be too tedious to resolve.

IV. TASTE OF KOTLIN

A new modern programming language, KOTLIN is much suited for all kinds of Multiplatform applications in today's world.

This language is well suited to supercharge your Android development. Kotlin replaces the very old JAVA which suffers from the "Billion Dollar Mistake" i.e. NULL POINTER EXCEPTION, the main cause for Android Applications crashes. To be said precisely it finds its usage widely instead of JAVA almost everywhere today .KOTLIN stands for it's - 100% interoperability with JAVA, application reliability , pragmatic nature, rich availability of standard library, safe, tool friendliness, concise and the list goes on and on. We shall now explore Kotlin's main features in a detailed manner .

V. SIGNIFICANT FEATURES OF KOTLIN

1. Null Safety
Nullable types and Non-Null Types
Kotlin's type system eliminates the danger of null references from code, also called as the ***The Billion Dollar Mistake***.
The type system helps us to differentiate between references that can hold null and those that cannot.
As an example, consider a regular variable of type String which cannot hold null:

```
var a: String = "abc"
a = null // compilation error
```

We can declare a variable as nullable string, written String? : to allow nulls

```
var b: String? = "abc"
b = null // ok
```

2. Java Interop
Both Java and Kotlin files can be used within the same project. In Java projects we can use Kotlin libraries and vice versa. The usage of Java and Kotlin classes simultaneously in the project eliminates the need for full project conversion or having to start a project from the beginning. A key factor behind the rising adoption rate of Kotlin is it's Interoperability with Java .
In a natural way existing Java code can be called from Kotlin, and rather smoothly as well Kotlin code can be used from Java. We shall describe some details about calling Java code from Kotlin.

```
import java.util.*
fun demo(source: List<Int>) {
val  list = ArrayList<Int>()
for(item in source) {list.add(item)}
for(i in 0..source.size - 1) {
list[i] = source[i] }}
```

The above code shows that Kotlin behaves very well in the existing Java ecosystem in terms of the usage of Java libraries, provides Java APIs, and integration with Java frameworks.
Kotlin mostly through extensions but sometimes with compiler-supported techniques (primitives, arrays, collections) relies on Java libraries. This gets us compatibility while keeping the language clean.

3. Checked Exceptions
The compiler in Kotlin does not force you to catch any of the exceptions which means all exceptions are unchecked. So, Kotlin does not force you to do anything when you call a Java method that declares a checked exception,:

```
fun render(list: List<*>, to: Appendable) {
for (item in list) {to.append(item.toString()) }}
```

4. Object Methods
In Kotlin all the references of the type java.lang.Object are turned into Any when Java types are imported into Kotlin. Because Any is not platform-specific, it declares only equals() , hashCode() and toString() as its members, so to make other members of java.lang.Object available, Kotlin uses extension functions.

5. Translation of type nothing
Since the type Nothing has no natural counterpart in Java it is special. Infact, every reference type in Java, accepts null as a value, and nothing doesn't even accept that. So in the Java world this type cannot be accurately represented. Hence Kotlin includes a raw type where an argument of type Nothing is used:

```
fun emptyList(): List<Nothing> = listOf()
// is translated to
// List emptyList() { ... }
```

6. Speed of Compilation
With tests revealing that Java is still the 'faster' language – at an average, ~13% faster compilation speeds (with Gradle) than Kotlin (14.2 seconds vs 16.6 seconds). Now for partial builds with incremental compilation, the speed advantage of Java more or less disappears, with Kotlin either being at par, or even just a bit faster,.
To be more precise clean builds are perfomed more quickly in Java but the two languages are alike for partial builds.

7. Execution Time & Verbosity
Generally the volume of coding in Java is considerably greater than in Kotlin. To be said Kotlin is a more compact lanaguage . The Kotlin Android Extensions,
facilitates references to be quickly imported inside Activity files (within Views) – and then, as part of the Activity that View can be used in the code which means the coders need not re-write the 'findViewByld' method for each case.Since this feature is not seen in Java the amount of boilerplate code is a lot greater than that in Kotlin. To use the Kotlin extensions simply add an additional plugin in the build.gradle file. The third-party dependency injection tools are not at all needed in Kotlin unlike Java.

Automatic conversion of Java to Kotlin
The Java to Kotlin converter integrated with IntelliJ helps us save a huge amount of time. It saves you from retyping the code. Migrating code from Java to Kotlin without it would take much longer.

7. Conciseness
Concise and Expressive syntax is the one of the major selling points of Kotlin . This is accomplished with the four ways discussed below.

a. Data Classes
Without much additional functionality sometimes we create classes which act simply as data containers in Java. For example consider the Address class that contains all data associated with a particular address:

```
public class Address {
    private String street;
    private String city;
    private int streetNumber;
    private String postCode;
    private Country country;   }
…
```

The above class is less searchable and less readable. So generate an immutable Address class which contains only getters but no setters pretty fast with a modern IDE.Here of Kotlin's major advantage is readability Kotlin's code is much clearer.

Now let's have a look at the similar class in Kotlin:

```
 data class Address(var street: String,
                  var streetNumber: Int,
                  var postCode: String,
                  var city: String,
                  var country: Country)
```

Instead we shall now create an immutable
data class using the val keyword instead of var and ensure that all the objects you pass in are immutable as well:

```
 data class Address(val street: String,
                  val streetNumber: Int,
                  val postCode: String,
                  val city: String,
                  val country: Country)
```

In Kotlin, var creates mutable variables whereas val is used to create immutable variables.

b. Smart Casts
There are often many situations in Java, where we often have to cast objects in the place of the compiler which could actually do this for us because it's clear that the object can be cast. Consider the following example:

```
 public class Cast {
    static void printString(String str) {
       System.out.println(str);     }

    public static void main(String[] args) {
       Object hello = "Hello, World!";
       if (hello instanceof String) {
          printString((String) hello);}  } }
```

The Java compiler would throw us an error if we have tried to change printString((String) hello) to just printString(hello because hello is of type Object.
The compiler thus could prove us that a actual parameter is hello for the formal parameter , str of printString(String str):

```
fun printString(str: String) {
    println(str)  }

fun main(args: Array<String>) {
    val hello: Any = "Hello, World!"
    if (hello is String) {
        printString(hello)     } }
```

Any type in Kotlin is the equivalent of Java's Object, similar to "is" is the equivalent of instanceOf, and that enables us to create package-level functions in Kotlin .This is known as Smart Casts in Kotlin.
. Whenever the compiler can prove that typecasting an object is safe appropriately, it will:

```
 // Smart cast #1
if (hello !is String) return
printString(hello)  // Smart cast

// Smart cast #2
if (hello is String &&hello.first().isLetter()) {   // Smart cast after &&
    println("The string starts with a letter")
 }
```

In smart cast #1, it is obvious that hello must be a string. Otherwise due to the return statement in the preceding line the control flow wouldn't even reach that point.
Smart casts #2 make use of lazy evaluation.

c.Functional Programming
Kotlin comes with functional capabilities baked in much similar to Java 8 which introduced functional language elements such as lambda expressions (function literals),. Look at the usage of function literals in Java 8:

```
public static void main(String[] args) {
    List<String> genres = Arrays.asList("Action", "Comedy", "Thriller");
    List<String>myKindOfMovies = genres.stream().filter(s ->s.length() > 6).map(s -> s + " Movie").collect(Collectors.toList());
    System.out.println(myKindOfMovies);    // Output: [Thriller Movie]  }
```

Kotlin makes this even easier. By convention, Kotlin creates an implicit parameter called "it" so that you can skip typing the parameter for lambda expressions with only one parameter,:

```
fun main(args: Array<String>) {
    val genres = listOf("Action", "Comedy", "Thriller")
    val myKindOfMovies = genres.filter{ it.length> 6 }.map { it + " Movie" }
    println(myKindOfMovies)
 }
```

Here there is actually another convention here. Whenever the last parameter of a method is a lambda expression, we can put it behind the parentheses of the method call.
In the previous example, both lambda expressions are the only parameters so that you can skip the parentheses altogether.
Also, the myKindOfMovies variables now stores an Iterable<String> there is no need for us to use a collect() method at the end,. This can also be made explicit if we want to:

```
val myKindOfMovies: Iterable<String> = genres.filter{ it.length> 6 }.map { it + " Movie" }
```

d. Objects as Singletons
To create a singleton though there are many ways, let's look at the most common way to create it in Java:

```
public class Singleton {
    private static Singleton theInstance= new Singleton();

    private Singleton() {     }

    public static Singleton getInstance() {
       returntheInstance;     } }
```

A class with a private constructor was created such that we can control which instances of it are created. Because we would like to have only one object of this class at any point of time, we shall instantiate it as a private attribute and allow retrieving it from the outside using getInstance().

By using the object keyword in Kotlin, we can create such a singleton in a single line:
object KotlinSingleton {}
Just like you may do it for a singleton in Java such object declarations in Kotlin are initialized lazily,.
This approach can nearly even keep up with the conciseness of Kotlin.

CONCLUSION

Kotlin makes Android programming easy to learn and lot more interesting.Though it has some of the drawbacks, it helps us to save ample amount of time in a much neater and helpful way.

I hope this paper gave you some more understanding of the Kotlin programming language and how it handles some of the more annoying aspects of Java in more convenient ways. Kotlin’s syntax is generally rather concise which makes your code more readable and thus maintainable.