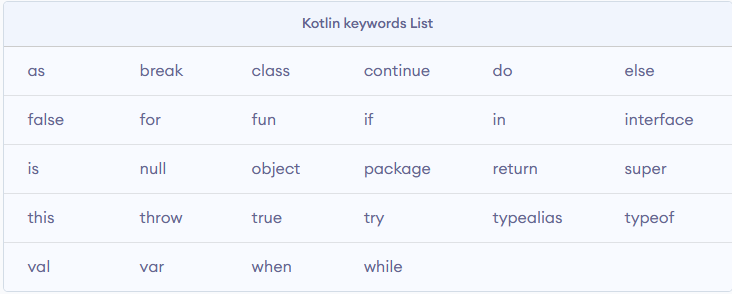
My chosen language: Kotlin

**Keywords**



Here is just a portion of some of the keywords in Kotlin. Many of the key words here are very easily readable and writeable. They all work exactly how you would think they would work based on the naming. It is actually very similar to Java. Because of this, it carries much of the fundamentals of readability and writability that Java has. For example, the as keyword is for casting. This makes sense since you would create an object “as” something. This makes reading the code easier. Also having keywords like typeof all one word is helpful too. It can make it feel like you are writing just regular pseudocode. Purely regarding the keywords, there are not really many problems besides maybe the reservation of the word itself. But, there are a subset of keywords called soft keywords which means they are considered keywords sometimes and sometimes not. This can end up with confusion in readability and writability since it can become hard to tell which is which.

**Data Type**

Data types in Kotlin are again similar to Java; however, the biggest thing is the fact that you can make something nullable or not by adding a question mark. For example, if you want a nullable String you can have String?, if you want a non-null String you can just use String. This will reserve some errors like NullPointerExceptions depending on the work that you are doing. Also another benefit is that you can use var and the compiler will figure out the type. Now this itself will somewhat hurt the readability of the code as the only the coder will know what was intended, but it will help the writability as the coder can seamlessly create variables in stride. Another big thing is the instanceOf method. If you were to check the instanceOf the datatype or object then within the if block, that variable will treated like it is of that. Because Kotlin uses var unlike Java’s strict data type naming system, it also makes creating ‘final’ variables easy. Instead of using var, you will just need to use val. This makes the variable ‘final’. All of this makes writing easier. The visual cues like the question mark and the instanceOf methods make it somewhat easier to read as well. Now, for some problems. Kotlin has something called Data classes. This is basically like a premade Java pojo. Imagine one line for creating a pojo java class – that’s what Kotlin can do. The problem therein lies when you try to use inheritance with it. Also when creating a class with all the different data types, it is by default considered final. This means that the writability is decreased since you will have to create dynamic code by writing the keyword ‘open’ at those classes.

**Control Structures**

There are primarily 4 control structures – if, when, for, while

I will compare them to java and explain the readability and writability in this way.

For the if statement, the readability and writability is very similar to Java. It is simply an if statement with some case next to it. But the neat thing in Kotlin is that you can actually use it as a block meaning you can assign it to a variable like so –

val max = if (a > b) {

print("Choose a")

a

} else {

print("Choose b")

b

}

In this form; however, there must be an else block as the value of the variables needs to have a guaranteed assignment. This can be considered a problem. This does increase the readability and writability in both ways because you can clearly see the options for the variable and the coder can easily continue coding instead of separating these details out.

The when statement is basically the same as the switch statement in Java, but with less wording. The when statement looks as such –

when (x) {

1 -> print("x == 1")

2 -> print("x == 2")

else -> { // Note the block

print("x is neither 1 nor 2")

}

}

You can see here that the writability is increased as you do not need to physically type as much since the need for naming cases is left out. It is also fairly readable.

Next, the for loop. The for loop in this is similar to the foreach in Java. This means though that the loop needs to loops over something that will allow an iterator. Normally this wouldn’t be a problem, as that is something you would normally loop over, but in the chance that it does not happen, you will need to find a work around. The readability for this is increased as the wording is as follows – for( item in collection) do something.

The while and do while, is the same as Java basically.

**Expressions**

There is support for operator overloading; however, this can serve as problem because if not handled correctly you could receive unwanted values. In Kotlin, there is not much support for unary operators. Hence, the allowing of the overloading. You could do something like the following

data class Point(val x: Int, val y: Int)

operator fun Point.unaryMinus() = Point(-x, -y)

val point = Point(10, 20)

fun main() {

println(-point) // prints "Point(x=-10, y=-20)"

}

This will provide you with some unary operators, but again this can result in many user errors if done incorrectly.

Binary operators are very similar to java with regards to ++ and – as well as anything like a + b. Now, the trinary operators also do not have much support in Kotlin. Therefore, the readability and writability is compared to that of Java.

Assignment

Assignment in Kotlin is almost like a double-edged sword. On one hand because creating final variable and assignable variables is the same character length (var vs val) it makes the writability easier, but the readability goes down as it can hard to distinguish between var and val easily. Also, because Kotlin compiler will interpret the variable type automatically, it can hard to read what kind of variable it is. Writing it does become easier though in that way. Then again, if you for sure want a variable to be int then you have to include var and int in the same line in order for it to be assigned as such which will decrease the writability.

Logic

The logic is very similar to Java; however if you dive deeper it can get complex as it will contain certain proprietary logic functionalities such as lambda expressions. This just makes the readability and writability decrease.

Order of operations

Same as logic, the order of operations is the same as Java, this does not really affect the readability and writability as it is pretty similar to all languages overall anyway.

Lastly, the overall reliability of Kotlin is okay. There is not much support for it, and because of that many developers can find themselves stuck in some situations. Also some of the things that were mentioned before can make Kotlin less reliable as well. For example creating data classes that cannot use inheritance does not help the reliability, as it will not help with the overall production of the code. Unfortunately, creating dynamic code in Kotlin is what is difficult because of the automatic final classes. It can be hard to inherit or really change classes quickly when altering the logic.

Sources:

<https://kotlinlang.org/docs/reference/keyword-reference.html>

https://www.programiz.com/kotlin-programming/keywords-identifiers