

## **Introdução ao desenvolvimento de aplicativos móveis Android com a linguagem Kotlin: um comparativo com Java.**

Autoria: Felipe Luiz Seixas e Silva

E-mail: [felipe.economista@hotmail.com](mailto:felipe.economista@hotmail.com)

LinkedIn: <https://www.linkedin.com/in/felipe-seixas>

### **Resumo**

O artigo descreve, através de estudo de caso, uma introdução em como criar um aplicativo mobile para a plataforma Android, de forma nativa, utilizando a linguagem de programação Kotlin (recomendada pelo Google INC como a linguagem prioritária, desde 2019). Identificando semelhanças e diferenças entre as implementações em Java (mais comum e antiga no mercado mobile) para contribuir positivamente na adaptação dos profissionais programadores(as) que possuem conhecimento prévio na linguagem de programação Java. Para isso foi desenvolvido um aplicativo, por meio da IDE Android Studio, que calcula e compara os custos de combustível de acordo com o destino informado. Foram abordados temas sobre: variáveis, condicionais, controles de fluxo, classes, funções e conversões de tipo. Como metodologia foi realizada a pesquisa exploratória e descritiva. E os delineamentos de pesquisa bibliográfica, documental e estudo de caso. O trabalho caracteriza-se como uma pesquisa qualitativa.

Palavras-chave: Kotlin; Mobile; Android; Java.

### **Introdução**

Durante o evento Google I/O 2019 a linguagem de programação Kotlin foi recomendada como prioritária para o desenvolvimento de aplicativos móveis, nativos, voltados para o sistema operacional Android. Sendo uma mudança de paradigma, pois antes a linguagem Java era a primeira opção do Google Inc, que é proprietária desse sistema operacional.

E como a predominância nesse mercado é o uso de Java, então esse artigo descreve e exemplifica os passos iniciais em Kotlin num projeto de aplicativo, comparando com o uso entre as duas linguagens.

Nesse contexto, notou-se a indagação: um(a) programador(a) Java encontrará diferenças de conceitos e sintaxe na construção de aplicativos com Kotlin?

Este artigo tem como objetivo descrever, através de estudo de caso (projeto de aplicativo na IDE Android Studio), uma introdução para a criação de aplicativos utilizando a linguagem Kotlin. Visando facilitar, ou seja, reduzir o tempo e o esforço

empregados na adaptação ao uso dessa linguagem. Principalmente, para quem possui conhecimento prévio em programação Java. Por meio de elucidação comparativas com as semelhanças e diferenças de conceitos/implementações entre as duas linguagens.

O tema abordado justifica-se pelo aumento da demanda de programadores(as) Java em reduzir a curva de aprendizado no desenvolvimento de aplicativos mobile com códigos Kotlin. Os desenvolvedores necessitam avaliar as semelhanças e diferenças entre as linguagens que atuam no dia a dia com as novas. Principalmente quando as duas linguagens podem ser usadas até mesmo juntas num mesmo projeto.

A metodologia deste artigo é do tipo exploratória e descritiva, pela descoberta e esclarecimento dos conceitos e sintaxe de Kotlin para projetos Android; divididos em dois delineamentos, a saber: bibliográfico – através de pesquisa em livros, artigos científicos e apostilas sobre conceitos e implementação das linguagens. E estudo de caso – por meio do estudo aprofundado de um projeto real de aplicativo móvel (elaborado de forma didática em para facilitar a compreensão desse trabalho).

Os aplicativos móveis estão presentes no cotidiano de muitas pessoas, possibilitando o acesso a informações pessoais e empresariais, independente do momento ou local. Suas capacidades podem ser executadas enquanto os dispositivos estão em movimento. (INTRODUÇÃO, 2014)

A criação de aplicativos concentra-se nos sistemas operacionais Android (Google) e iOS (Apple). Para fins desse estudo será considerado o uso do sistema Android. Por deter o maior número de usuários, aproximadamente 90% do mercado, e por possuir o maior número de aplicativos disponíveis em sua loja virtual.

Já os dispositivos mobile tem como características: portátil, fonte de alimentação de energia própria, conectividade sem fio, capacidade computacional, processamento e armazenamento.

E dentre as variadas abordagens de desenvolvimento como: nativa, híbrida e webmobile. Este estudo terá como uso a forma “nativa” que caracteriza-se, de forma geral, como a codificação utilizando a linguagem de programação específica de uma plataforma.

A forma nativa é reconhecida como diferencial entre as demais formas de desenvolvimento pelo seu alto potencial de desempenho, usabilidade, experiência do usuário, segurança e acesso a todos os recursos (câmeras, bússola,

acelerômetro etc) do smartphone. E a ferramenta mais indicada é utilizando-se o Kit de Desenvolvimento de Software (SDK) disponibilizado pelo fabricante da plataforma. (INTRODUÇÃO, 2014)

E como fator menos favorável, está o maior custo de mão de obra (criação e manutenção). Outro fator é que a atualização de algumas funcionalidades não é transparente aos usuários, requerendo a visita deles à loja virtual para obter a versão mais recente. E por último, a obrigatoriedade de passar pelo processo de aprovação do aplicativo na loja que distribui, no caso do Android (objeto deste trabalho) é a Google Play.

A plataforma de desenvolvimento mais utilizada e que será a referência para o artigo é o Android Studio: “(...) é a IDE oficial de desenvolvimento para Android. O Android Studio foi anunciado no Google I/O 2013 e é baseado no IntelliJ IDEA da JetBrains. (SAMUEL; BOCUTIU, 2017, p.44)

De acordo com Developers,

O Android Studio é o ambiente de desenvolvimento integrado (IDE, na sigla em inglês) oficial para o desenvolvimento de apps Android e é baseado no [IntelliJ IDEA](#). Além do editor de código e das ferramentas de desenvolvedor avançadas do IntelliJ, o Android Studio oferece ainda mais recursos para aumentar sua produtividade na criação de apps Android (...). (DEVELOPERS, 2020)

Conforme destacado por Developers,

O Android Studio tem total compatibilidade com o Kotlin, assim você pode criar novos projetos com arquivos Kotlin, adicionar arquivos Kotlin ao seu projeto e converter código de linguagem Java em Kotlin. Você pode usar todas as ferramentas atuais do Android Studio com o código Kotlin, incluindo conclusão de código, verificação de lint, refatoração, depuração e muito mais. (DEVELOPERS, 2020)

Conforme David e Dawn, “O Android Studio é o IDE oficial do Android e o mais recomendado pela equipe de desenvolvimento da plataforma.” e “(...) oferece uma interface gráfica para o Gradle e outras ferramentas de criação de layouts, leitura de logs e depuração”. Suportando várias linguagens de programação, dentre elas: Java, C, C++ e Kotlin (mais recente). (DAVID; DAWN, 2019, p.07)

O uso da IDE Android Studio para a criação de aplicativos Android (nativo) é bem popular. Principalmente para os desenvolvedores mobile que atuam com a linguagem de programação Java.

Em se tratando dos conceitos sobre a linguagem de programação Kotlin, Segundo Samuel e Bocutiu,

Kotlin é uma linguagem JVM, portanto o compilador gera bytecodes Java. Por causa disso, é claro, o código Kotlin pode chamar o código Java e vice versa! Assim, será necessário ter o Java JDK instalado em sua máquina. Para ser capaz de escrever código para Android, no qual a versão mais recente aceita é o Java 6. O compilador deve traduzir seu código para bytecodes, que sejam, no mínimo, compatíveis com Java 6. (SAMUEL; BOCUTIU, 2017, p. 20)

E conforme identificado no site oficial da linguagem,

Kotlin é uma linguagem de programação moderna, mas já madura, destinada a tornar os desenvolvedores mais felizes. É conciso, seguro, interoperável com Java e outras linguagens e oferece muitas maneiras de reutilizar código entre várias plataformas para uma programação produtiva. (KOTLINLANG, 2022)

Sendo assim, a união de Kotlin e Android Studio permite criar soluções para dispositivos móveis diversos, como relógios, automóveis, braceletes e *Smartphones*. Este último será a referência principal para o dispositivo mobile neste artigo. Trata-se de um telefone celular com funcionalidades avançadas que podem ser estendidas por meio de programas executados por seu sistema operacional. Com diferentes tamanhos, modelos de hardware etc. (INTRODUÇÃO, 2014)

## **Desenvolvimento**

Como estudo de caso foi criado, via Android Studio (versão 4.0), um aplicativo chamado “KotlinCustoCombustível” que calcula e compara os custos de combustíveis (gasolina, álcool e gás veicular) de acordo com o destino informado (quilometragem) selecionando a linguagem Kotlin como padrão do projeto, conforme detalhes nos apêndices A e B.

A linguagem Kotlin trabalha com variáveis por meio de duas palavras reservadas, *val* e *var*: a) “*val*” é utilizada quando se deseja uma variável de leitura (figura 1, linhas 14 e 19). Devendo-se ser inicializada logo que criada, pois não pode ser modificada, diretamente, em outro momento, assemelhando-se com a variável final em Java. Podendo ser alterada apenas via instâncias, ou seja, através de funções ou propriedades; b) E a palavra reservada “*var*” (figura 1, linhas 15 e 18) remete-se a uma variável comum em Java, pois permite ter seu conteúdo alterado uma ou mais vezes após sua inicialização, na mesma instância, inclusive.

```

13 // Inteiros:
14 val pMedio: Int = 100
15 var precoMedio = 6
16
17 // Doubles:
18 var precoLocal = 7.10
19 val capacidadeTanque: Double = 40.4

```

Figura 1 – Variáveis

Essas duas formas de se declaram variáveis em Kotlin tem em comum a não exigência da declaração explícita do seu tipo. Pois o compilador busca associar o tipo de acordo com o conteúdo inicializado. Essa é uma diferença significativa para o Java. Tornando Kotlin menos verbosa nesse cenário de variáveis sem abrir mão de ser uma linguagem fortemente tipada (mantendo a segurança de tipos), porém implicitamente.

Sobre o conceito de igualdade entre variáveis, Kotlin faz uso da chamada “referencial” quando duas referências separadas apontando para o mesma instância de memória. Nesse teste utiliza-se a igualdade tripla “===” ou “!==” para negação.

Essa linguagem utiliza a igualdade “estrutural”, quando deseja-se saber se as instâncias separadas na memória possuem ou não o mesmos valores atribuídos naquele momento, via operadores “==” (verdade) ou “!=” (negação). E isso é uma notável diferença para Java, em que esses dois operadores são classificados como igualdade “referencial”, ou seja, o inverso.

No geral, os tipos básicos de Kotlin são todos objetos, ou seja, os tipos primitivos foram promovidos a objetos completos. Diferente de Java, que permite tipos primitivos e objetos num mesmo projeto. Sobre os tipos primitivos, de acordo com Samuel e Bocutiu, “(...) Esses tipos não podem ser usados como genéricos, não aceitam chamadas de métodos/funções e nem podem receber null (...)”. Como exemplo o uso do tipo primitivo booleano nas implementações em Java. (SAMUEL; BOCUTIU, 2017, p. 47)

Contudo, motivado em melhorar o desempenho, o compilador de Kotlin mapeará os tipos básicos(objetos) para primitivos na máquina virtual Java (JVM).

Os tipos numéricos básicos, conforme ilustrado na figura 2, são: *Long*, *Int*, *Short*, *Byte*, *Double* e *Float*. No projeto, as variáveis foram explicitamente tipadas para facilitar o entendimento da sintaxe, conforme figura 2, linhas 21-26.

```

21      val meuLong: Long = 100
22      var meuInt: Int = 100
23      val meuShort: Short = 100
24      var meuByte: Byte = 100
25      val meuDouble: Double = 40.4
26      var meuFloat: Float = 40.4F

```

Figura 2 – Tipos numéricos

O tipo *double* é o padrão para flutuantes. E caso necessário utilizar um valor do tipo *float*, então será preciso escrever o sufixo F (maiúsculo) ao final do valor desejado.

De acordo com Samuel e Bocutiu, a conversão de números é realizada explicitamente, pelas funções que cada tipo possui. Visto que todos os tipos dessa linguagem são objetos. São as funções: `toByte()`, `toShort()`, `toInt()`, `toLong()`, `toFloat()`, `toDouble()`, `toChar()`. Conforme dois exemplos da figura 3, entre inteiros (linha 40) e doubles (linha 38). (SAMUEL; BOCUTIU, 2017, p. 48)

```

38      novoDouble = meuInt.toDouble()
39
40      novoInt = meuDouble.toInt()

```

Figura 3 - Conversão de tipos numéricos

O tipo Booleano é idêntico ao conhecido em Java, com *true* e *false*. E aceitam os operadores comuns de negação, conjunção e disjunção. Esses dois últimos trabalham de forma otimizada, em que caso o teste do lado esquerdo satisfaça a cláusula, o teste do lado direito não será avaliado (conhecido como modo *lazy*).

Conforme exemplo da figura 4, linha 45, caso a variável “meuInt” seja maior que “meuDouble”, então a variável “verdadeLazy” é inicializada sem que o teste “meuLong < meuInt” seja realizado, pois o seu resultado não influenciaria na inicialização da variável com *true* ou *false*.

```

45      val verdadeLazy = meuInt > meuDouble || meuLong < meuInt
46      val verdade = meuInt > meuDouble && meuLong < meuInt

```

Figura 4 – Teste de condição (modo *lazy*)

O tipo *Char*, assemelha-se ao Java, por representar um único caractere. Mas diferem entre si, quando em Kotlin ele não pode ser tratado também como um número. Também apresenta os seguintes tipos de *escapes*: `\t`, `\b`, `\n`, `\r`, `'`, `"`, `\\` e `\$`.

Em se tratando de *Strings* há algumas diferenças importantes em relação ao Java. Uma novidade chama-se *templates* de *strings*, descrito como: “maneira simples e eficaz para incluir valores, variáveis ou até mesmo expressões em uma string”. Sem a necessidade de utilizar padrões ou concatenação de *strings* para combinar expressões com *strings* literais, prática comum em Java. (SAMUEL; BOCUTIU, 2017, 52)

Sua sintaxe é reconhecida pelo uso do símbolo de cifrão (\$) junto e pré-fixado a uma variável ou um valor (figura 5, linha 49). Além de permitir a utilização de expressões arbitrárias, nesses casos iniciando com o uso de chaves ({}), junto ao cifrão (\$) (figura 6, linha 51).

Segundo Samuel e Bocutiu, os templates “(...) representam uma melhoria em relação à experiência com Java, quando múltiplas variáveis são utilizadas em um único literal”. (SAMUEL; BOCUTIU, 2017, p.53)

Para as *strings* literais são utilizadas as aspas duplas(“ ”), no qual permitem o uso de escapes (figura 5, linha 54), mas como novidade, Kotlin possui as aspas triplas (“””) para gerar uma *string* bruta, em que todos os caracteres entre as aspas são considerados como caracteres, sendo assim são armazenados em sua totalidade, mesmo digitando escapes eles serão considerados como caracteres, conforme observa-se na figura 5, linha 56. E como exemplo mais simples, sem escape, na figura 5, linha 58. Nesse exemplo foram expostas os resultados da execução do código Kotlin no terminal da IDE, visando facilitar o entendimento:

```

// Template de strings
val titulo = "calculadora"

val personalizado = "Bem vindo(a)! $titulo iniciada."

val contaTitulo = "O título do aplicativo possui ${titulo.length} caracteres."

// Strings
val combEscapeAtivo = "\n 1) Gasolina; \n 2) Alcool; e \n 3) Gas veicular."
val combEscapeInativo = """\n 1) Gasolina; \n 2) Alcool; e \n 3) Gas veicular."""
val msgFinal = """Obrigado por utilizar, compartilhe! \n"""
    
```

Logcat

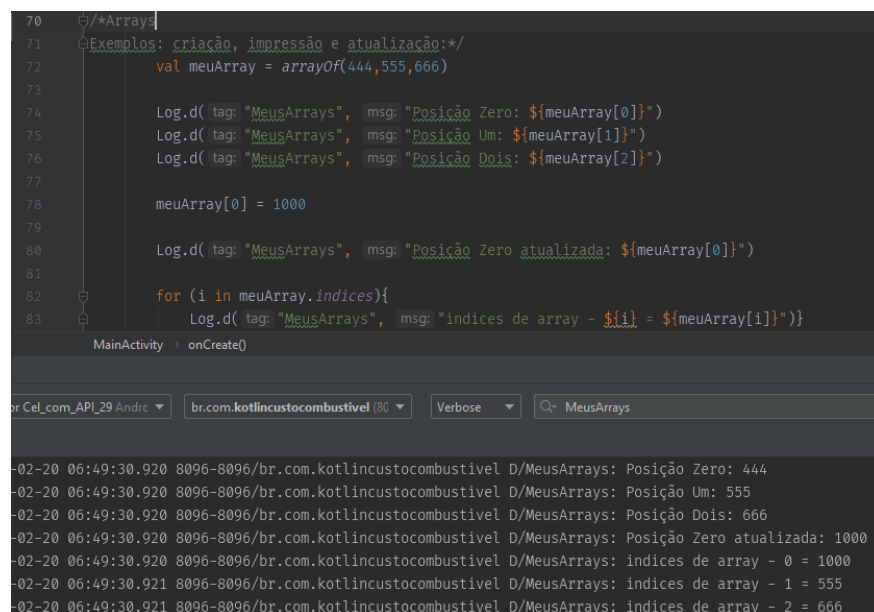
```

2022-02-17 13:01:01.049 3227-3227/br.com.kotlincustocombustivel I/string20: Bem vindo(a)! Calculadora iniciada.
2022-02-17 13:01:01.049 3227-3227/br.com.kotlincustocombustivel I/string21: O título do aplicativo possui 11 caracteres.
2022-02-17 13:01:01.050 3227-3227/br.com.kotlincustocombustivel D/string22: Aspas duplas (com escape):
1) Gasolina;
2) Alcool; e
3) Gas veicular.
2022-02-17 13:01:01.050 3227-3227/br.com.kotlincustocombustivel D/string23: Aspas triplas (com escape): \n 1) Gasolina; \n 2) Alcool; e \n 3) Gas veicular.
2022-02-17 13:01:01.050 3227-3227/br.com.kotlincustocombustivel D/string24: Aspas triplas: Obrigado por utilizar, compartilhe! \n
    
```

Figura 5 – *Strings*

De forma prática, caso de uso comum seria na alternância de telas (activity's) dos aplicativos, passando e recebendo *strings* como parâmetro, por exemplo. Ou para formas de controle de logins, armazenando de tokens e/ou similares em strings.

Outro tema importante para iniciar em Kotlin é o uso de *arrays*, podemos utilizá-los pela função “`arrayOf( )`”. Os *arrays* são tratados como classes comuns de coleção, diferente de Java que os considera como especiais. Por padrão, a biblioteca oferece as funções: *Interação*, *Size*, *Get* e *Set*.



```

70  /*Arrays
71  Exemplos: criação, impressão e atualização:*/
72      val meuArray = arrayOf(444,555,666)
73
74      Log.d( tag: "MeusArrays", msg: "Posição Zero: ${meuArray[0]}")
75      Log.d( tag: "MeusArrays", msg: "Posição Um: ${meuArray[1]}")
76      Log.d( tag: "MeusArrays", msg: "Posição Dois: ${meuArray[2]}")
77
78      meuArray[0] = 1000
79
80      Log.d( tag: "MeusArrays", msg: "Posição Zero atualizada: ${meuArray[0]}")
81
82      for (i in meuArray.indices){
83          Log.d( tag: "MeusArrays", msg: "índices de array - ${i} = ${meuArray[i]}")
84      }
85  }
86  MainActivity - onCreate()

```

Log Output:

```

02-20 06:49:30.920 8096-8096/br.com.kotlincustocombustivel D/MeusArrays: Posição Zero: 444
02-20 06:49:30.920 8096-8096/br.com.kotlincustocombustivel D/MeusArrays: Posição Um: 555
02-20 06:49:30.920 8096-8096/br.com.kotlincustocombustivel D/MeusArrays: Posição Dois: 666
02-20 06:49:30.920 8096-8096/br.com.kotlincustocombustivel D/MeusArrays: Posição Zero atualizada: 1000
02-20 06:49:30.920 8096-8096/br.com.kotlincustocombustivel D/MeusArrays: índices de array - 0 = 1000
02-20 06:49:30.921 8096-8096/br.com.kotlincustocombustivel D/MeusArrays: índices de array - 1 = 555
02-20 06:49:30.921 8096-8096/br.com.kotlincustocombustivel D/MeusArrays: índices de array - 2 = 666

```

Figura 6 – Arrays

Como ferramenta de desempenho a linguagem Kotlin procura evitar o boxing de tipos, oferecendo alternativas de arrays, especializada para cada um dos tipos primitivos. São elas: a) *ByteArray*, *CharArray*, *Short Array*, *Int Array*, *Long Array*, *Boolean Array*, *Float Array* e *Double Array*.

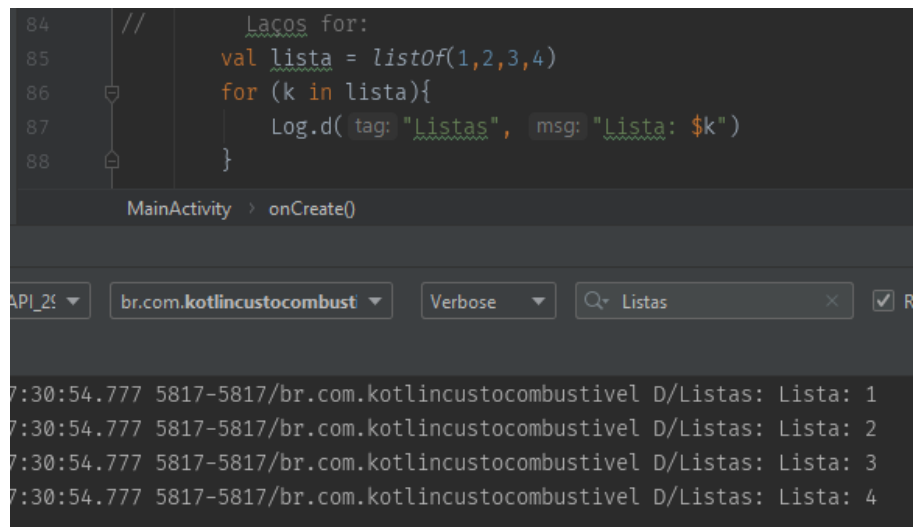
O `Array.indices` é uma função que pode ser utilizada para iterar pelo índice do array, como exemplificado na figura 6, linhas 82 e 83.

No quesito comentários, nota-se igualdade entre as duas linguagens. Sendo “//” para comentário de linha (figura 5, linhas 46 e 53) e “/\*...\*/” voltado para os de bloco (figura 6, linhas 70-71).

Em Kotlin, para laços de controle são utilizados “*while*” e “*for*”, assim como em Java. O laço *for* “( )” é usado para iterar por qualquer objeto que defina uma função ou uma função de extensão com o iteradores de nome. Todas as coleções disponibilizam essa função.” (SAMUEL, BOCUTIU, 2017, p.54).



É conhecido que os laços utilizam também como referência os intervalos (ranges), assim como em Java, Kotlin os representam com um valor “inicial” até outro “final” (figura 7). Contudo, há uma nova forma de personalizar esse intervalo: via operador “...” respeitando a condição de utilizar qualquer tipo comparável (números, caracteres etc), permitindo, inclusive, testes de “está contido” nesse intervalo. Sendo maior ou igual ao valor inicial; ou menor igual ao valor final.



The screenshot shows an IDE with Kotlin code in a file named MainActivity.kt. The code is as follows:

```
84 // Laços for:
85 val lista = listOf(1,2,3,4)
86 for (k in lista){
87     Log.d( tag: "Listas", msg: "Lista: $k")
88 }
```

Below the code editor, the execution path is shown as MainActivity > onCreate(). The bottom panel displays the log output for the application:

```
7:30:54.777 5817-5817/br.com.kotlincustocombustivel D/Listas: Lista: 1
7:30:54.777 5817-5817/br.com.kotlincustocombustivel D/Listas: Lista: 2
7:30:54.777 5817-5817/br.com.kotlincustocombustivel D/Listas: Lista: 3
7:30:54.777 5817-5817/br.com.kotlincustocombustivel D/Listas: Lista: 4
```

Figura 7 – Listas utilizano o operador “in”

Além do operador “...”, uma novidade, em relação ao Java, é o operador “in” (figuras 7 e 8). Podendo ser utilizado em laços “for” para facilitar o uso de intervalos, quer seja para percorrer coleções ou mesmo intervalos inteiros (figura 8, linha 91), pré-definidos ou ainda mais simples, escrevendo-se na linha/definidos externamente (figura 8, linha 93).

Também há a opção de realizar iterações, baseando-se na quantidade de caracteres de uma *string*, por exemplo na figura 8, linhas 99-100.

```

90 //      Uso do controle "in":
91      val umAdez = 1..10
92      for (k in umAdez){
93          for (j in 1..5){
94              //      Log.d("ControleIn", "Multiplicação: ${k*j}")
95          }
96      }
97 //      Uso do "in" para contar caracteres de uma string:
98      val combustivel = "Gás"
99      for (char in combustivel){
100          Log.d( tag: "ControleIn", msg: "In com string: $char")
101      }
102
MainActivity > onCreate()

```

Logcat output:

```

:11:37.692 5817-5817/br.com.kotlincustocombustivel D/ControleIn: In com string: G
:11:37.692 5817-5817/br.com.kotlincustocombustivel D/ControleIn: In com string: á
:11:37.692 5817-5817/br.com.kotlincustocombustivel D/ControleIn: In com string: s

```

Figura 8 - Interação com contagem de caracteres

Visando a otimização do desempenho para os laços e *arrays*, os intervalos são tratados de modo especial pelo compilador, sendo compilados para laços “*for*” baseados em índices e, diretamente, aceitos na JVM, evitando, assim, qualquer penalidade no desempenho, proveniente da criação de objetos iteradores (SAMUEL, BOCUTIU, 2017, p.55)

Em Kotlin, quando tratamos sobre as instâncias de classes, não deverá ser utilizada a palavra “*new*” seguida do nome da classe a ser criada (pois ela não é uma palavra reservada nesta linguagem, comum em Java, e caso seja escrita ocorrerá um erro de compilação).

Programando em Kotlin a chamada de uma função de construção ocorre igualmente a uma chamada comum de uma função normal, usando diretamente o uso da classe desejada, de forma direta e objetiva.

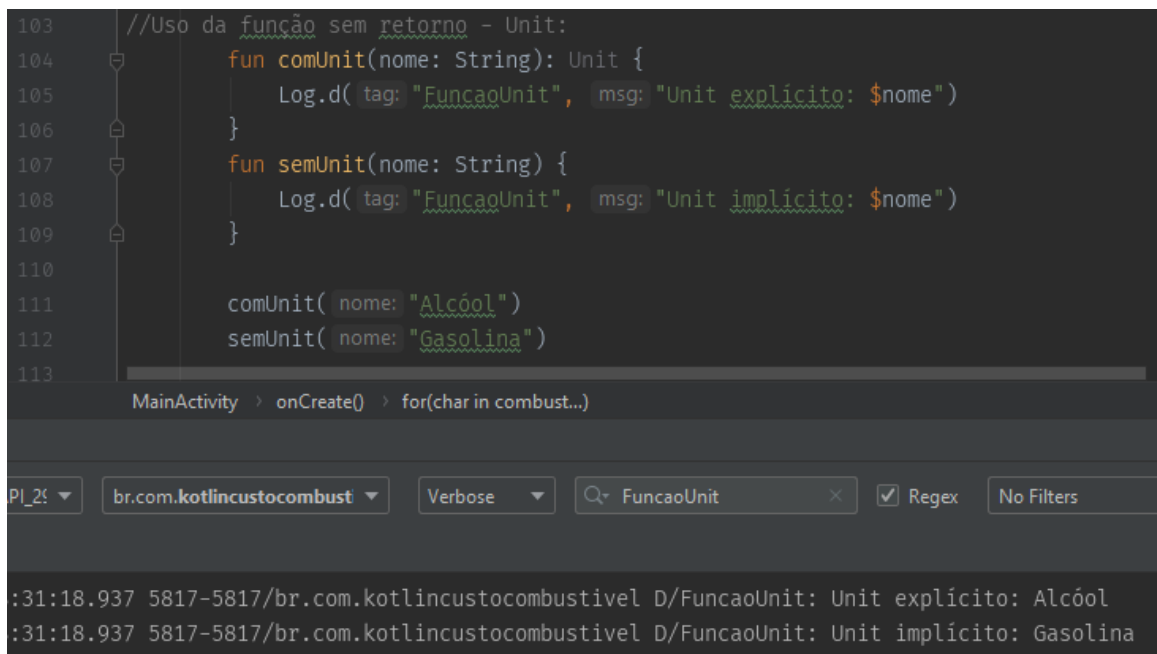
Sobre o controle de fluxo, segundo Samuel e Bocutiu: “os blocos de controle *if..else* e *try..catch* são expressões, isso significa que o resultado pode ser diretamente atribuído a um valor, devolvido por uma função ou passado como argumento para outra função.” (SAMUEL, BOCUTIU, 2017, p.61)

Obrigatoriamente, ao utilizar a cláusula *if* é preciso ter uma cláusula *else* correspondente, caso contrário ocorrerá um erro de compilação em tempo de execu-

ção, diferente de Java em que não ocorre qualquer erro de compilação, seja em tempo de execução ou não.

Em Kotlin a programação funcional (uso de funções) inicia pelo uso da palavra reservada “*fun*”, parâmetros (não obrigatórios) e um valor de retorno. Com a lista de parâmetros sempre presente, mesmo quando não houver qualquer parâmetro definido.

Os parâmetros devem respeitar o formato nome: tipo. E quando a função não devolver qualquer valor, então deve ser definida como *Unit*, palavra reservada semelhante ao efeito *void*, em Java. Essa função também aceita omitir o tipo de retorno na sintaxe, conforme os exemplos equivalentes (figura 9).



```

103 //Uso da função sem retorno - Unit:
104 fun comUnit(nome: String): Unit {
105     Log.d( tag: "FuncaoUnit", msg: "Unit explícito: $nome")
106 }
107 fun semUnit(nome: String) {
108     Log.d( tag: "FuncaoUnit", msg: "Unit implícito: $nome")
109 }
110
111 comUnit( nome: "Alcôol")
112 semUnit( nome: "Gasolina")
113
    MainActivity > onCreate() > for(char in combust...)

```

PL\_2! br.com.kotlincustocombust Verbose Q FuncaoUnit [x] [x] Regex No Filters

```

:31:18.937 5817-5817/br.com.kotlincustocombustivel D/FuncaoUnit: Unit explícito: Alcôol
:31:18.937 5817-5817/br.com.kotlincustocombustivel D/FuncaoUnit: Unit implícito: Gasolina
  
```

Figura 9 - Funções Unit (sem retorno)

Outra função em Kotlin é conhecida como “*on line*” ou “*single line*” (figura 10), ela é mais curta na sua escrita (sintaxe) e suas três principais características são: a) A possibilidade da omissão das chaves; b) A não declaração do valor de retorno (Int, Char etc), pois é inferido pelo compilador; e c) Exigir o acréscimo do símbolo “=” antes da expressão, em vez de utilizar a palavra reservada “*return*”.

Conceitualmente, segundo Samuel e Bocutiu, “(...) funções bem pequenas são fáceis de ler e o valor de retorno é um pequeno ruído extra que não acrescenta muito ao processo como um todo.” (SAMUEL, BOCUTIU, 2017, p.114)

```

114 // Função "single line":
115 fun calculaLitros (distancia: Double, consumo: Double) = distancia / consumo
116
117 val totalLitros = calculaLitros( distancia: 40.0, consumo: 14.5)
118 Log.d( tag: "FuncaoSingle", msg: "Serão necessários : $totalLitros litros para concluir o trajeto.")
119
120 }

```

MainActivity > onCreate()

br.com.kotlincustocombustivel (58) Verbose Q FuncaoSingle [x] [x] Regex No Filters

43:27.217 5817-5817/br.com.kotlincustocombustivel D/FuncaoSingle: Serão necessários : 2.7586206896551726 litros para concluir o trajeto.

Figura 10 - Funções “in line”

Outra novidade em Kotlin são os “parâmetros nomeados”, que permitem explicitar os significados dos argumentos passados numa função. Para isso, escreve-se o nome do parâmetro antes do valor do argumento.

Sendo indicado para funções que possuem muitos parâmetros, escrevendo-se mais, contudo deixando o código de mais fácil entendimento. Evitando mais chances de se consultar a documentação do código fonte. Assim como é vantajoso para situações em que há poucos parâmetros, mas sendo do mesmo tipo, reduzindo as chances de erros por trocas.

Nem todos os parâmetros precisam ser nomeados, mas quando isso ocorrer é preciso ter atenção, pois, obrigatoriamente, os demais (a direita) deverão ser nomeados também. É permitido, ainda, mudar a ordem dos parâmetros sem afetar o resultado. (SAMUEL, BOCUTIU, 2017)

Em Java é comum o uso da instrução switch, mas em Kotlin essa funcionalidade tem outra instrução/sintaxe chamada “when” (figura 11). Ela apresenta-se de duas formas: com ou sem argumentos.

Na primeira (figura 11, linhas 122-129), o argumento deve estar entre parênteses e permite testar várias condições, no qual são verificadas baseando-se no valor passado como argumento. Ela aceita muitas validações juntas, separando-os por “vírgula”, quando for o caso.

E quando as validações não encontram o argumento pode-se utilizar a palavra reservada “else” com a instrução desejada, seguindo o mesmo conceito do “default” dentro do switch em Java.

```
120 // When com argumento:
121 fun informaDescontos(x: Int){
122     when(x){
123         1 -> Log.d( tag: "Descontos", msg: "Desconto baixo")
124
125         2 -> Log.d( tag: "Descontos", msg: "Desconto moderado")
126
127         3 -> Log.d( tag: "Descontos", msg: "Desconto alto")
128
129         else -> Log.d( tag: "Descontos", msg: "Sem desconto")
130     }
131 }
132 //
133 fun melhorPrecoPostos(postoUm: Double, postoDois: Double){
134     when{
135         postoUm < postoDois -> Log.d( tag: "Preços", msg: "Abasteca no posto 1")
136
137         postoUm > postoDois -> Log.d( tag: "Preços", msg: "Abasteca no posto 2")
138
139         else -> Log.d( tag: "Preços", msg: "Os preços estão iguais")
140     }
141 }
```

Figura 11 - Instrução when

Já a segunda forma não possui argumentos, sendo usada quando há uma necessidade de testa/comparar mais de uma validação, conforme exemplo na figura 11, linhas 134-139, como forma de evitar vários “if...e/se”, buscando assim maior legibilidade dos testes/validações.

## Considerações finais

O artigo permitiu conhecer, através de estudo de caso, a descrição das semelhanças e diferenças conceituais/sintaxe, entre as linguagens de programação Kotlin e Java ao criar um aplicativo móvel para Android (nativo). Com potencial para facilitar a adaptação dos profissionais programadores(as) que possuem conhecimento prévio na linguagem de programação Java. Com isso, foram apresentados vários exemplos reais de códigos em Kotlin.

Foi possível elaborar um projeto de aplicativo para calcular e comparar os custos de combustíveis (gasolina, álcool e gás veicular). E a partir deste, identificar conceitos, a sintaxe e interpretar os códigos relacionados a temas importantes na descoberta de uma nova linguagem, como: variáveis, tipos básicos, templates de *strings*, *arrays*, índices, laços de controle *for* e *while*; operadores de intervalos, testes *if...else*, funções e a instrução de controle *when*.

Com isso, a partir deste artigo o profissional de programação móvel Android poderá dispor de menos tempo e esforço para iniciar, de forma prática, a criação de aplicativos com Kotlin, Podendo servir de orientação para quem deseja entrar ou permanecer nesse mercado de desenvolvimento móvel. Já que essa linguagem é a prioritária do Google Inc, proprietária do sistema Android.

E como limitações, o artigo não abrange a forma como os *layouts* e as *views* são manipulados via código Kotlin, além de não abordar mais temas e conceitos necessários para projetos mais abrangentes e complexos. Logo, para suprir essa necessidade, recomendam-se aprofundar os estudos em gerenciamento de *layouts* e *views*, classes, modificadores de visibilidade, polimorfismo, herança e encapsulamento; segurança com nulos, genéricos e coleções.

## Referências

DAVID, Griffiths; DAWN, Griffiths. **Use a cabeça! Desenvolvendo para Android**. 2 Ed. São Paulo: Alta Books, 2019.

DEVELOPER. **Primeiros passos com o Kotlin no Android**. Disponível em: <<https://developer.android.com/kotlin/get-started>>. Acesso em 13 fev 2022.

\_\_\_\_\_. **Conheça o Android Studio**. Disponível em: <<https://developer.android.com/studio/intro>>. Acesso em 13 fev 2022.

DEVELOPERS, Google. **Devo aprender a usar Kotlin para Android e outras perguntas frequentes**. Disponível em: <<https://developers-br.googleblog.com/2020/11/devo-aprender-usar-o-kotlin-para.html>>. Acesso em 5 fev 2022.

**INTRODUÇÃO ao desenvolvimento nativo**. (s.l:s.n), 2014. 31 páginas. Apostila.

KOTLINLANG. **Introdução ao Kotlin**. Disponível em: <<https://kotlinlang.org/docs/getting-started.html>>. Acesso em 13 fev 2022.

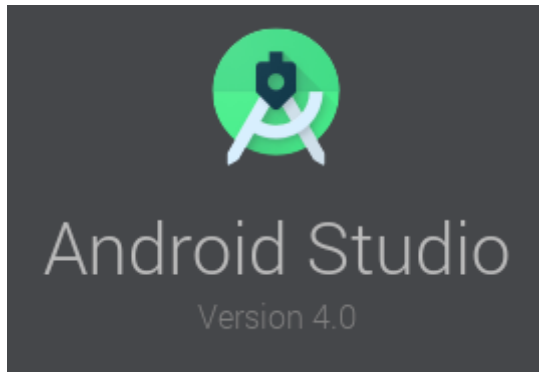
\_\_\_\_\_. **Variáveis**. Disponível em: <<https://kotlinlang.org/docs/basic-syntax.html#variables>>. Acesso em: 13 fev 2022.

LAKATOS, Eva Maria; MARCONI, Marina de Andrade. **Fundamentos de metodologia científica**. 5. ed. rev. ampl. São Paulo: Ed. Atlas, 2010.

LECHETA, Ricardo. **Google Android: aprenda a criar aplicações para dispositivos móveis com Android SDK**. 5. Ed. São Paulo: Novatec, 2016.

SAMUEL, Stephen; BOCUTIU, Stefan. **Programando com Kotlin: conheça todos os recursos de Kotlin com este guia detalhado**. São Paulo: Novatec, 2017.

## APÊNDICE A – IDE utilizada e a respectiva versão



## APÊNDICE B – Nome do projeto e a linguagem utilizada

