

Fundamentos de Desenvolvimento Android

MIT em Desenvolvimento Mobile - 2020

Sintaxe Básica da Linguagem Kotlin

Linguagem Kotlin

Em julho de 2011, a JetBrains revelou o Projeto Kotlin - uma nova linguagem para o JVM.

Um dos objetivos da linguagem Kotlin é compilar tão rapidamente quanto o Java.

Em fevereiro de 2012, a JetBrains abriu o código do projeto sob a licença Apache 2.

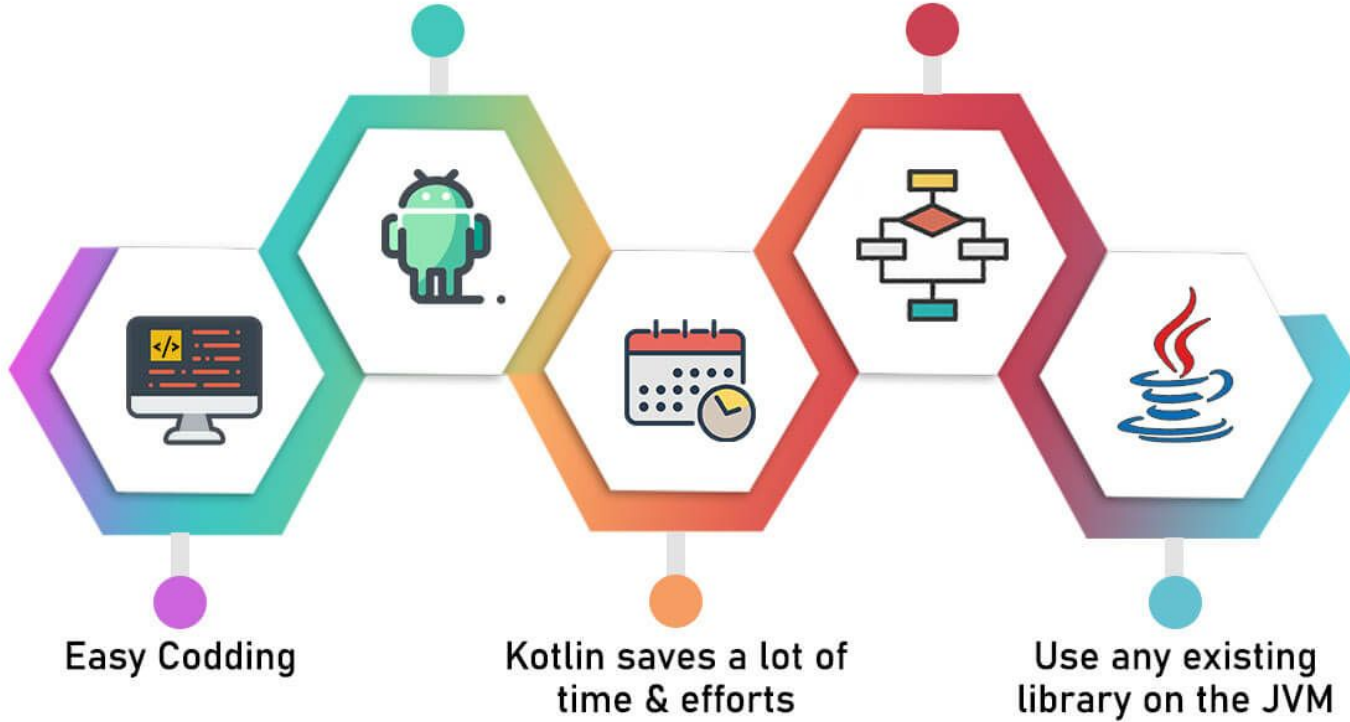
Kotlin 100% interoperável com Java TM e Android TM.



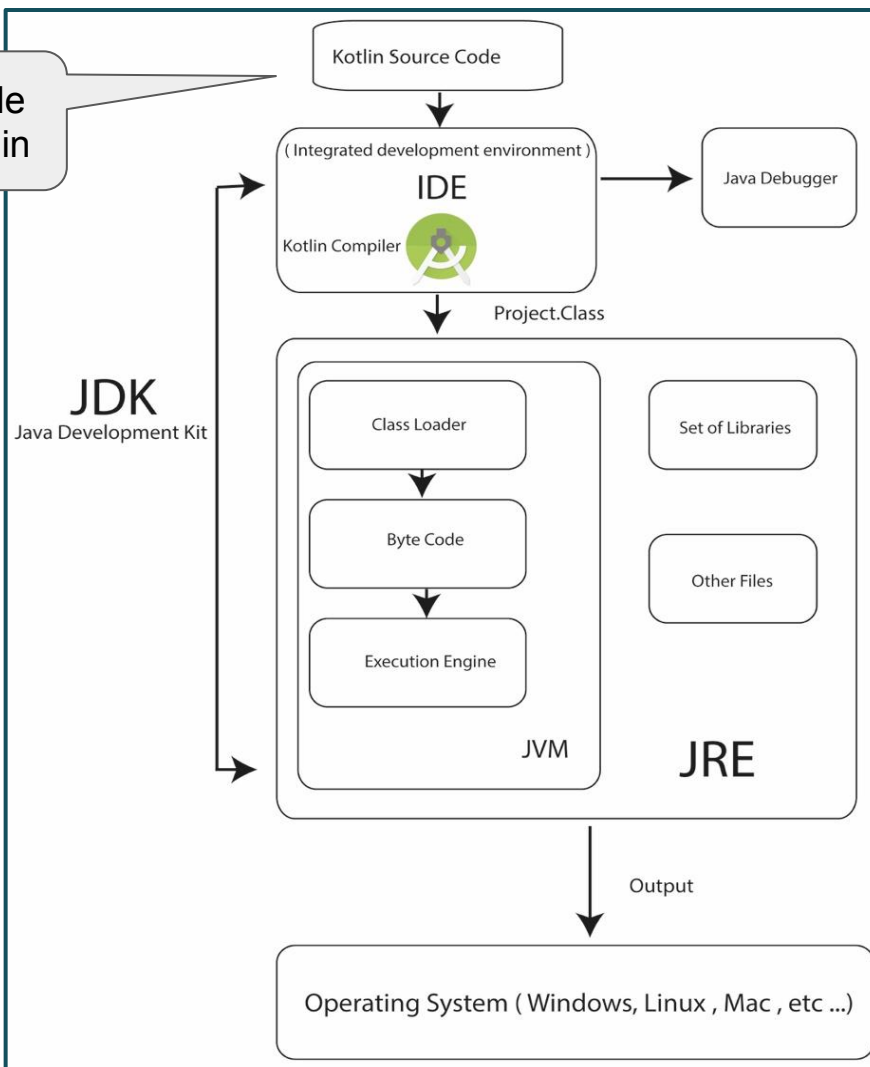


Android Studio Support

Best Procedural
Programming



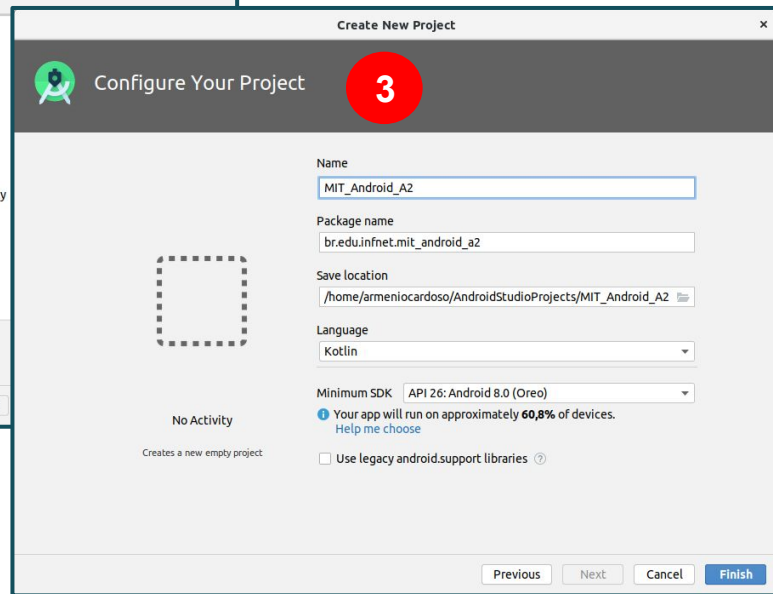
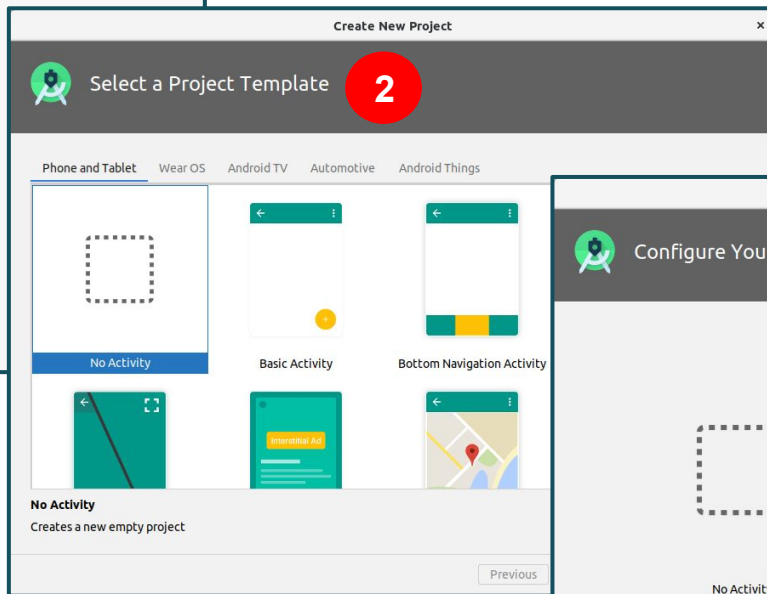
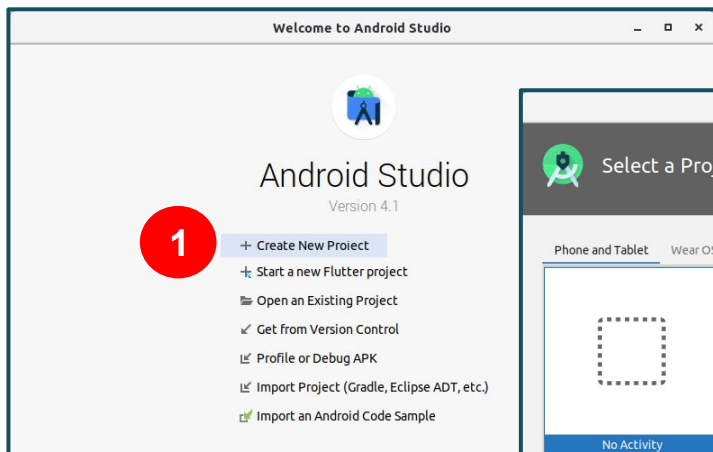
Ciclo de vida de
um código Kotlin



Linguagem Kotlin

No diagrama você pode observar que para escrever um código Kotlin deve instalar o IntelliJ IDEA ou Android Studio como IDE e para executar o Java JDK.

Primeiro Programa



Primeiro Programa



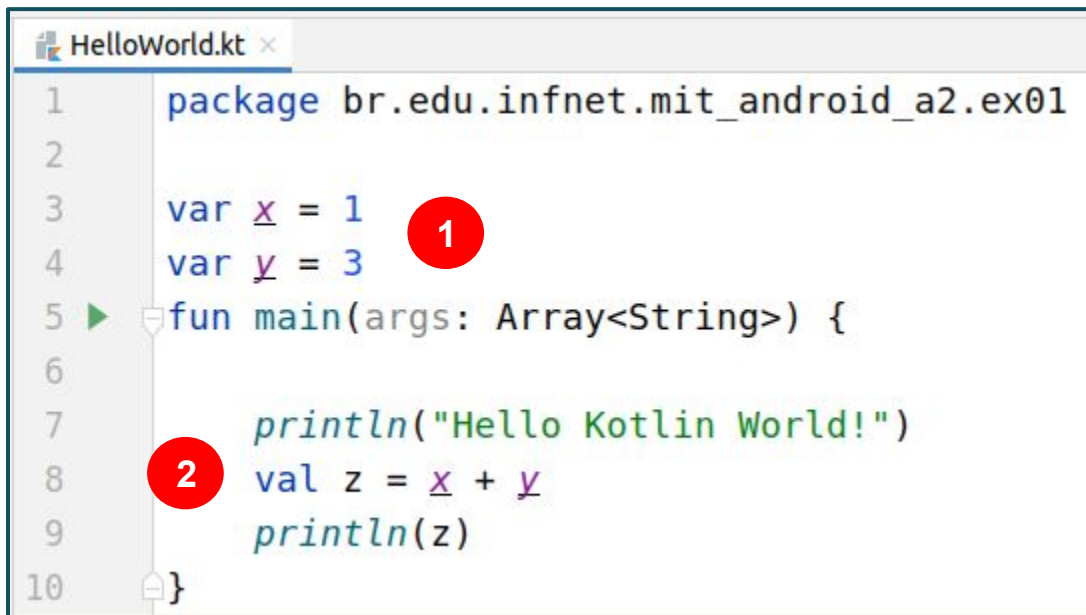
```
1 package br.edu.infnet.mit_android_a2.ex01
2
3 fun main(args: Array<String>) {
4
5     println("Hello Kotlin World!")
6 }
```

Run: HelloWorldKt

```
/snap/android-studio/94/android-studio/jre/bin/java ...
Hello Kotlin World!
Process finished with exit code 0
```

Variáveis → **var** X **val**

Uma variável é aquela cujo valor pode ser alterado a qualquer momento. Para declarar uma variável, escrevemos **var** diretamente antes da variável. O exemplo a seguir dá uma ideia de como usar uma variável:



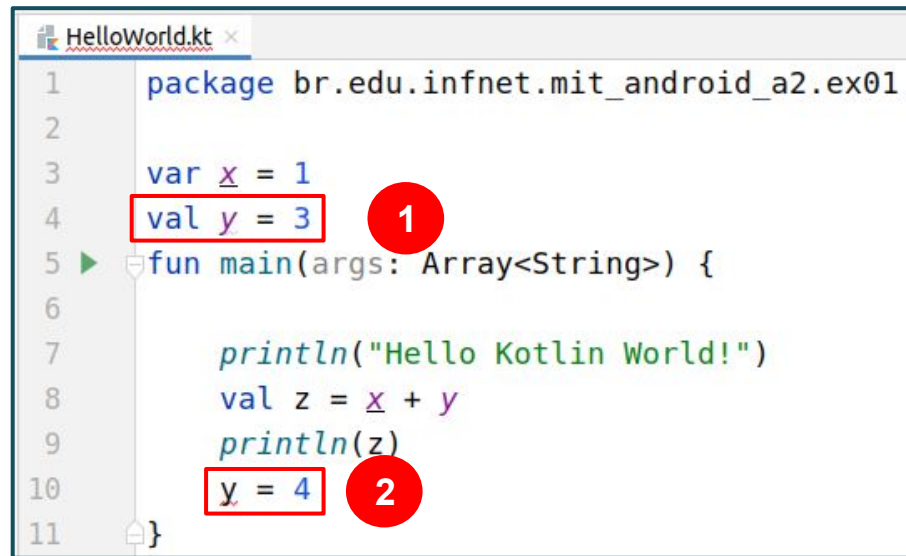
```
1 package br.edu.infnet.mit_android_a2.ex01
2
3 var x = 1
4 var y = 3
5 fun main(args: Array<String>) {
6
7     println("Hello Kotlin World!")
8     val z = x + y
9     println(z)
10 }
```

The screenshot shows a code editor window titled "HelloWorld.kt". The code is written in Kotlin. Line 3 declares a variable `x` with the value 1. Line 4 declares a variable `y` with the value 3. Line 5 starts the `main` function. Line 7 prints "Hello Kotlin World!". Line 8 declares a `val` variable `z` as the sum of `x` and `y`. Line 9 prints the value of `z`. Line 10 ends the function. Two red circles with numbers are overlaid on the image: circle 1 is next to the declaration of `x` on line 3, and circle 2 is next to the declaration of `z` on line 8.

Constantes → var X val

Uma constante não pode ser alterada após a inicialização. Para declarar uma constante, escrevemos **val** diretamente antes da declaração.

A imagem a seguir exibe o erro sublinhado em vermelho para a constante **y** quando tentamos alterar seu valor:



```
1 package br.edu.infnet.mit_android_a2.ex01
2
3 var x = 1
4 val y = 3
5 fun main(args: Array<String>) {
6
7     println("Hello Kotlin World!")
8     val z = x + y
9     println(z)
10    y = 4
11 }
```

Instrução **lateinit**

lateinit significa inicialização tardia.

Se você não deseja inicializar uma variável no construtor, em vez disso, deseja inicializá-la mais tarde e se pode garantir a inicialização antes de usá-la, então declare essa variável com a palavra-chave **lateinit**.

Ele não alocará memória até que seja inicializado.

Facilita muito na criação de variáveis sem ter que “inventar” valores iniciais ou ter que usar nulo.



Instrução **lateinit**

Você não pode usar **val** para a variável **lateinit**, pois ela será inicializada posteriormente.

Além disso, você deve garantir que está inicializando a variável antes de usá-la, caso contrário, ela lançará uma exceção em tempo de execução.

Você não pode usar **lateinit** para propriedades de tipo primitivo como **Int**, **Long** etc.



Tipos de Dados → String

Em Kotlin o tipo de uma variável ou constante é inferido pelo literal que é atribuído.

Mas também é possível e às vezes necessário declarar formalmente o tipo do dado.

```
12      //-----  
13      val escritor = "Machado de Assis"  
14      println(escritor)  
15      val escritora : String = "Rachel de Queiroz"  
16      println(escritora)  
17      //-----
```



Tipos de Dados → Char

O tipo de dados **Char** é usado para armazenar um único caractere como 'a', 'Z', '&', '\$', '8' e assim por diante.

Os caracteres podem ser letras ou símbolos especiais. Os caracteres são sempre colocados entre aspas simples, enquanto as aspas duplas são para Strings.

Se você deseja armazenar o valor '\$' em uma variável x, você deve escrever a seguinte declaração:

```
17 //-----  
18 val dolar: Char = '$'  
19 println(dolar)  
20 //-----
```

Tipos de Dados → Boolean

Um tipo de dados booleano possui dois valores possíveis: **true** ou **false**.

Os booleanos são usados em declarações de tomada de decisão que você pode controlar no fluxo do programa.

```
20      //-----  
21      var flag: Boolean = true  
22      flag = false  
23      println(flag)  
24      //-----
```

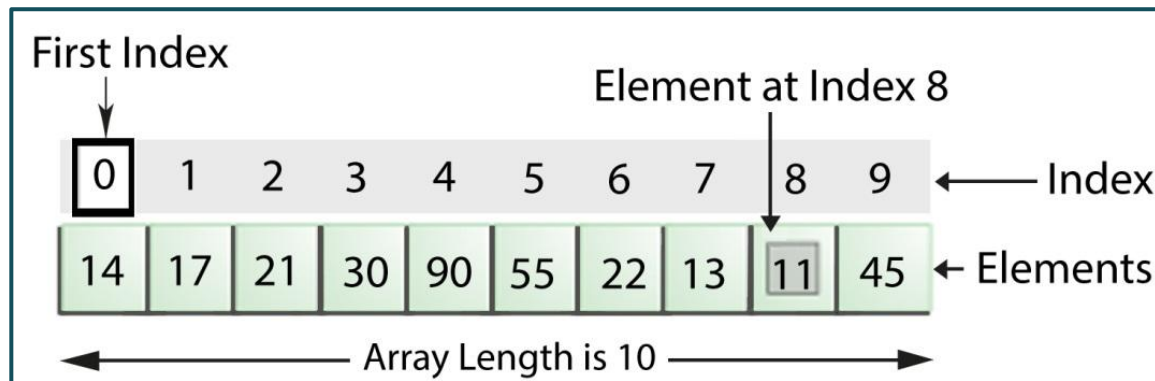
Tipos de Dados → Números

Data Type	Description	Default Value
Byte	The byte data type is an 8-bit signed integer. Its value is from -128 to 127	0
Short	The short data type is a 16-bit signed integer. Its value ranges from - 32768 to 32767	0
Int	The integer data type is a 32-bit signed integer. It has values from -2,147,483,648 to +2,147,483,647	0
Long	The long data type is a 64-bit signed integer.	0L
Float	The float data type is a single-precision 32-bit floating point.	0.0f
Double	The double data type is a double-precision 64-bit floating point.	0.0d

Tipos de Dados → Array

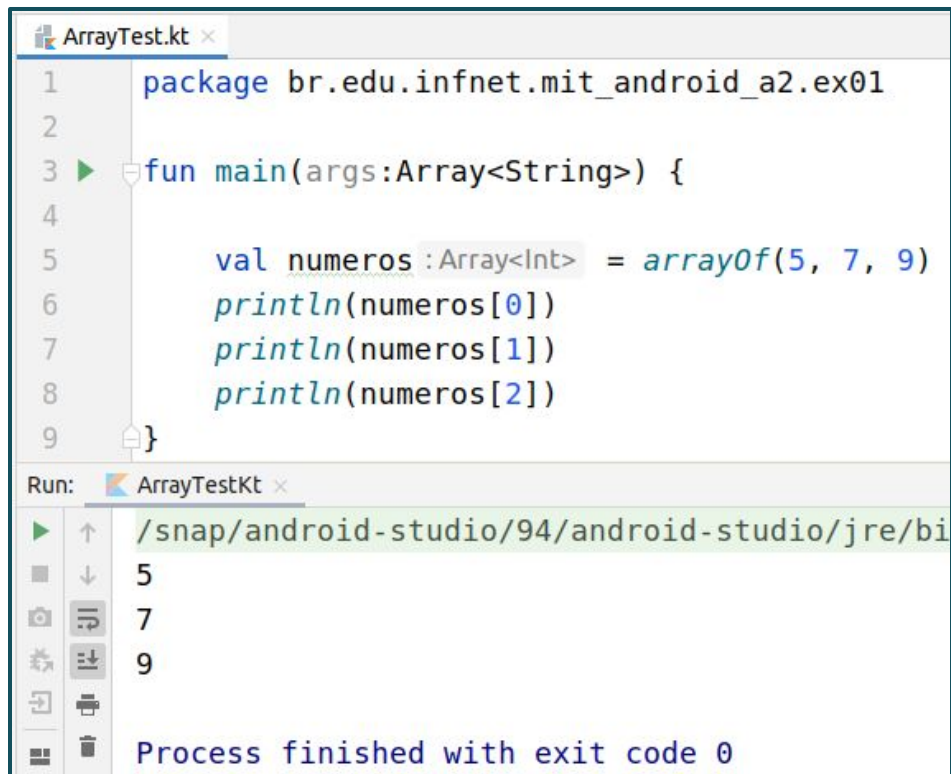
Se você deseja armazenar um grande número de itens de dados para a mesma variável, você precisa usar um vetor.

Um vetor é usado para armazenar um grupo de valores, todos com o mesmo tipo de dados.



Tipos de Dados → Array

Para criar um array, podemos usar uma função de biblioteca `arrayOf()` e passar os valores dos itens para ela, de modo que um array de (5, 7, 9) cria um array [5, 7, 9].



```
ArrayTest.kt x
1 package br.edu.infnet.mit_android_a2.ex01
2
3 fun main(args:Array<String>) {
4
5     val numeros :Array<Int> = arrayOf(5, 7, 9)
6     println(numeros[0])
7     println(numeros[1])
8     println(numeros[2])
9 }
```

Run: ArrayTestKt x

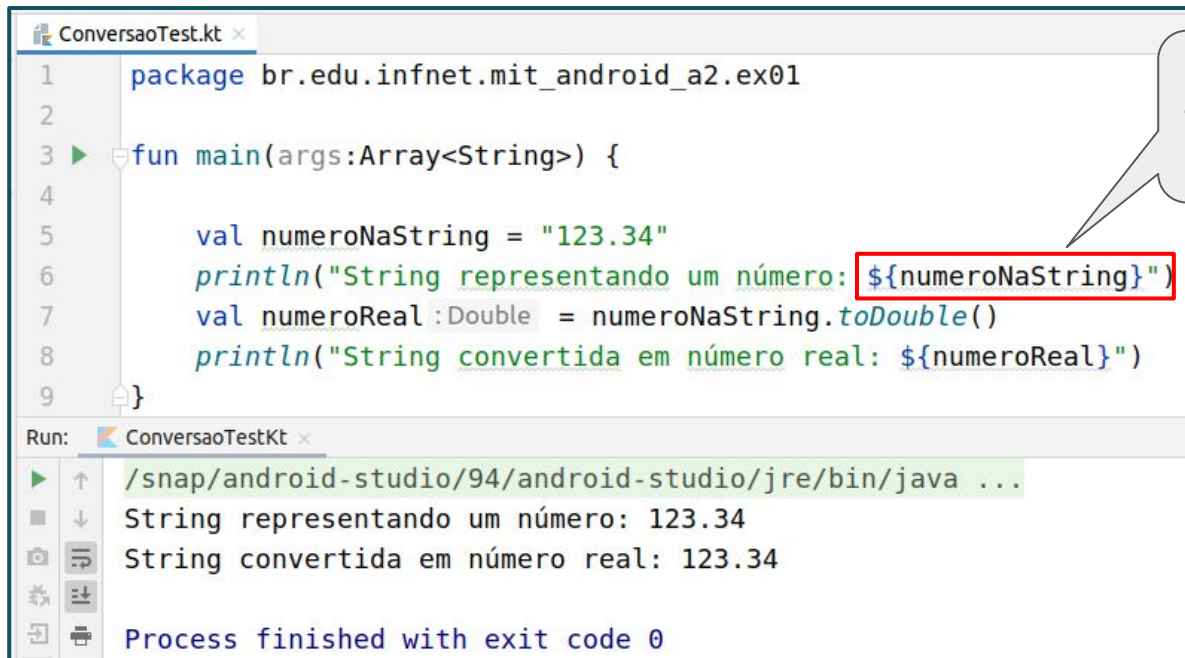
```
/snap/android-studio/94/android-studio/jre/bi
5
7
9

Process finished with exit code 0
```

Tipos de Dados → Conversões de Dados

Em alguns casos, você pode precisar converter um tipo de dados de uma variável em outro tipo de dados, como alterar um inteiro para um short.

- toByte()
- toShort()
- toInt()
- toLong()
- toFloat()
- toDouble()
- toChar()



```
ConversaoTest.kt x
1 package br.edu.infnet.mit_android_a2.ex01
2
3 fun main(args:Array<String>) {
4
5     val numeroNaString = "123.34"
6     println("String representando um número: ${numeroNaString}")
7     val numeroReal:Double = numeroNaString.toDouble()
8     println("String convertida em número real: ${numeroReal}")
9 }
```

Run: ConversaoTestKt x

```

1 /snap/android-studio/94/android-studio/jre/bin/java ...
2 String representando um número: 123.34
3 String convertida em número real: 123.34
4
5 Process finished with exit code 0
```

String
template - \$
ou \${}

Tipos de Dados → Conversões de Dados

```
ConversaoTest.kt x
1 package br.edu.infnet.mit_android_a2.ex01
2
3 fun main(args:Array<String>) {
4
5     val numeroNaString = "123.34"
6     println("String representando um número: ${numeroNaString}")
7     val numeroReal:Double = numeroNaString.toDouble()
8     println("String convertida em número real: ${numeroReal}")
9     //-----
10    val numeroInteiro:Int = numeroNaString.toInt()
11    println("String convertida em número inteiro: ${numeroInteiro}")
12 }

Run: ConversaoTestKt x
/snap/android-studio/94/android-studio/jre/bin/java ...
String representando um número: 123.34
String convertida em número real: 123.34
Exception in thread "main" java.lang.NumberFormatException: For input string: "123.34"
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    at java.lang.Integer.parseInt(Integer.java:580)
    at java.lang.Integer.parseInt(Integer.java:615)
    at br.edu.infnet.mit_android_a2.ex01.ConversaoTestKt.main(ConversaoTest.kt:10)

Process finished with exit code 1
```

Operadores Aritméticos

```
AritmeticaTest.kt x
1 package br.edu.infnet.mit_android_a2.ex01
2
3 fun main(args:Array<String>) {
4
5     val x = 123
6     val y = 456
7     //-----
8     var sum :Int = x + y
9     var mult :Int = x * y
10    var div :Int = x / y
11    var mod :Int = x % y
12    println("Sum = $sum --- Mult = $mult --- Div = $div --- Mod = $mod")
13    //-----
14    sum =x.plus(y)
15    mult = x.times(y)
16    div = x.div(y)
17    mod = x.rem(y) //remain ---> resto
18    println("Sum = $sum --- Mult = $mult --- Div = $div --- Mod = $mod")
19 }
```

Operadores Condicionais

A tabela a seguir inclui alguns operadores condicionais que ajudam no fluxo de controle do programa Kotlin:

Operator	Name
==	Equal to
!=	Not Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

Operadores Lógicos

Operadores lógicos como OR “| |” e AND “&&” devem ser usados para a construção de expressões lógicas mais complexas.

Quando você tem várias condições (ou seja, várias expressões booleanas) os operadores lógicos são usados para testar mais de uma condição ao mesmo tempo.

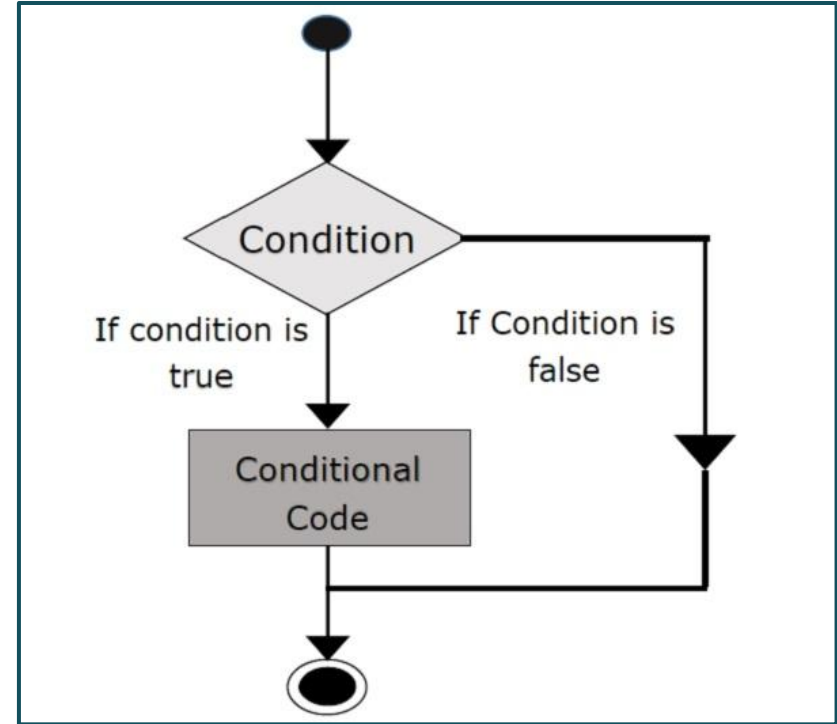
A	B	A B
true	true	true
true	false	true
false	true	true
false	false	false

A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false

Instrução **if**

A instrução **if** é a mais básica de todas as instruções de fluxo de controle.

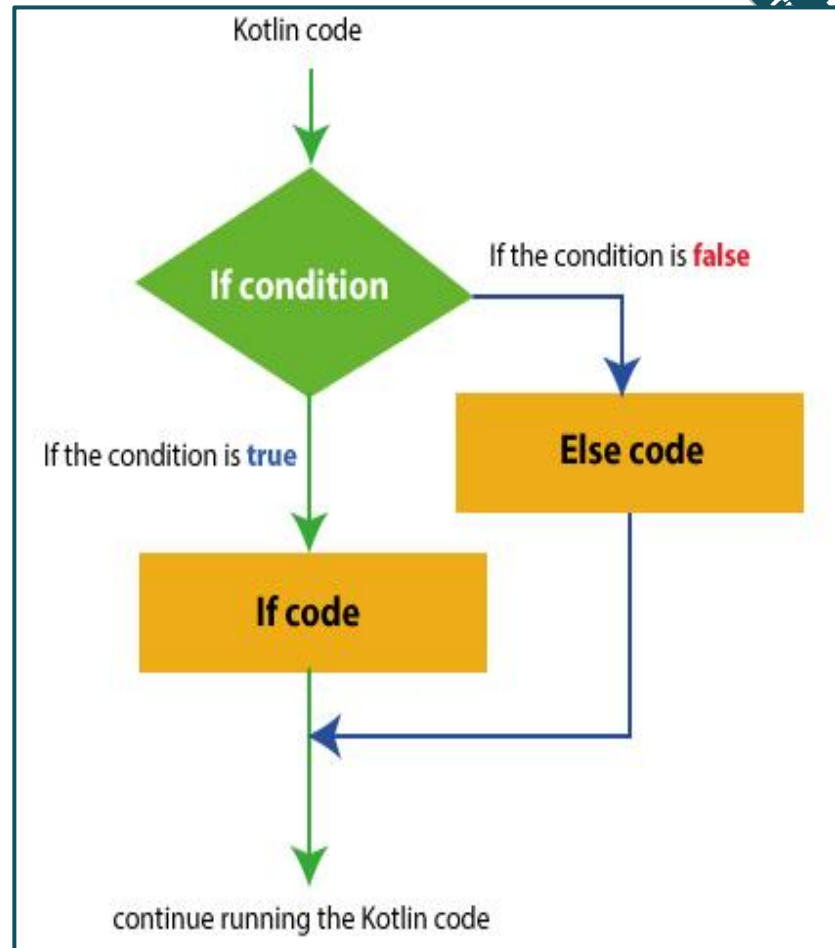
Diz ao seu programa para executar uma seção específica de um código apenas se um teste particular provar ser verdadeiro.



Instrução **if-else**

Se usar o **if** com a cláusula **else** existe o fluxo alternativo, caso a condição seja falsa.

Desejamos executar um código quando a condição for verdadeira e outro quando a condição for falsa.




```
CondicaoTest.kt x
1 package br.edu.infnet.mit_android_a2.ex01
2
3 fun main(args: Array<String>) {
4
5     var salario = 1234.56
6     var imposto = 0.0
7     if (salario > 1000 && salario <= 3000) {
8
9         imposto = salario * 0.05
10    } else if (salario > 3000 && salario <= 5000) {
11
12        imposto = salario * 0.2
13    } else {
14        imposto = salario * 0.27
15    }
16    println("Salário = $salario --- Imposto = $imposto")
17 }
```

Run: CondicaoTestKt x

▶ ↑ /snap/android-studio/94/android-studio/jre/bin/java ...

■ ↓ Salário = 1234.56 --- Imposto = 61.728

⚙ ⚙ Process finished with exit code 0

Instrução **when**

A instrução **when** é semelhante à instrução **switch** em outras linguagens de programação, como Java e C++.

A instrução **when** pode ter vários fluxos de execução possíveis.

Usando **when** podemos colocar três opções e, se nenhuma delas for verdadeira, o Android Studio executará a ação relacionada à instrução **else**.

A instrução **when** permite que você use muitas condições para a mesma variável.

```

WhenTest.kt x
1 package br.edu.infnet.mit_android_a2.ex01
2
3 import java.util.*
4
5 fun main(args: Array<String>) {
6
7     val diasSemana :Array<String> = arrayOf("Segunda", "Terça", "Quarta", "Quinta", "Sexta", "Sábado", "Domingo")
8     val reader = Scanner(System.`in`)
9     print("Escolha um dia da semana: 0-segunda, 1-terça, 2-quarta...6-domingo > ")
10    var diaEscolhido :Int = reader.nextInt()
11    when(diaEscolhido) {
12
13        0, 1, 2, 3, 4 -> println("${diasSemana[diaEscolhido]} é dia útil")
14        5, 6 -> println("${diasSemana[diaEscolhido]} é dia de descanso")
15        else -> println("Opção errada")
16    }

```

Run: WhenTestKt x

```

> /snap/android-studio/94/android-studio/jre/bin/java ...
Escolha um dia da semana: 0-segunda, 1-terça, 2-quarta...6-domingo > 4
Sexta é dia útil

Process finished with exit code 0

```



Instrução **for**

Uma instrução **for** fornece uma maneira compacta de iterar em uma faixa de valores.

Os programadores costumam se referir a ele como “for-loop” por causa de como ele faz loops repetidamente até que uma condição específica seja satisfeita.

```
WhenTest.kt
10 var diaEscolhido : Int = reader.nextInt()
11 when(diaEscolhido) {
12
13     0, 1, 2, 3, 4 -> println("${diasSemana[diaEscolhido]} é dia útil")
14     5, 6 -> println("${diasSemana[diaEscolhido]} é dia de descanso")
15     else -> {
16         println("Opção errada - as opções aparecem listadas abaixo")
17         for(diaSemana : String in diasSemana) {
18
19             println(diaSemana)
20         }
21     }
22 }
```

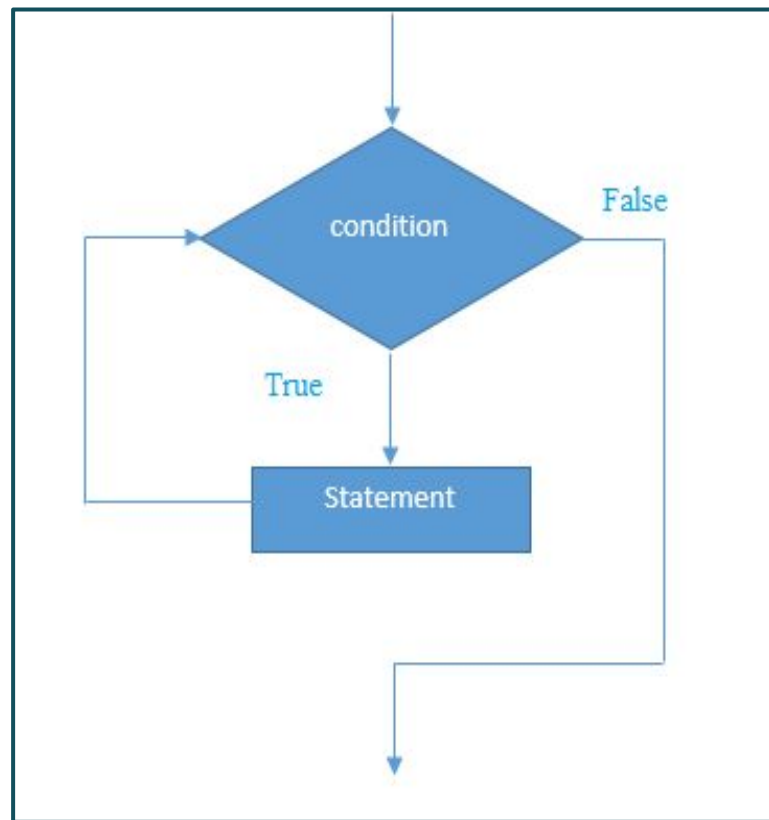
```
WhenTest.kt
10 var diaEscolhido : Int = reader.nextInt()
11 when(diaEscolhido) {
12
13     0, 1, 2, 3, 4 -> println("${diasSemana[diaEscolhido]} é dia útil")
14     5, 6 -> println("${diasSemana[diaEscolhido]} é dia de descanso")
15     else -> {
16         println("Opção errada - as opções aparecem listadas abaixo")
17         for(indice : Int in 0..diasSemana.size-1) {
18
19             println("$indice - ${diasSemana[indice]}")
20         }
21     }
22 }
```

Instrução **while**

Uma instrução **while** depende de uma expressão lógica.

Se a expressão provar ser verdadeira, a instrução **while** executa as instruções no bloco **while**.

A instrução **while** continua testando a expressão e executando seu loop até que a expressão seja provada como falsa.



WhenTest.kt x

```
5 fun main(args: Array<String>) {
6
7     val diasSemana :Array<String> = arrayOf("Segunda", "Terça", "Quarta", "Quinta", "Sexta", "Sábado", "Domingo")
8     val reader = Scanner(System.`in`)
9     var diaEscolhido = -1
10    while(diaEscolhido != 99) {
11
12        print("Escolha um dia da semana: 0-segunda, 1-terça, 2-quarta...6-domingo, 99-sair > ")
13        diaEscolhido = reader.nextInt()
14        when(diaEscolhido) {
15
16            0, 1, 2, 3, 4 -> println("${diasSemana[diaEscolhido]} é dia útil")
17            5, 6 -> println("${diasSemana[diaEscolhido]} é dia de descanso")
18            99 -> println("Saindo...")
19            else -> {
20                println("Opção errada - as opções aparecem listadas abaixo")
21                for(indice :Int in 0..diasSemana.size-1) {
22
23                    println("$indice - ${diasSemana[indice]}")
24                }
25            }
26        }
27    }
28 }
```

Instrução **break**

A instrução **break** permite que você saia de um loop em qualquer ponto e ultrapasse sua expressão de término normal.

Quando a instrução **break** é encontrada dentro de um loop, o loop é encerrado imediatamente e retoma seu trabalho a partir da instrução que segue o loop.

A instrução **break** pode ser usada em qualquer tipo de loop.

Instrução **continue**

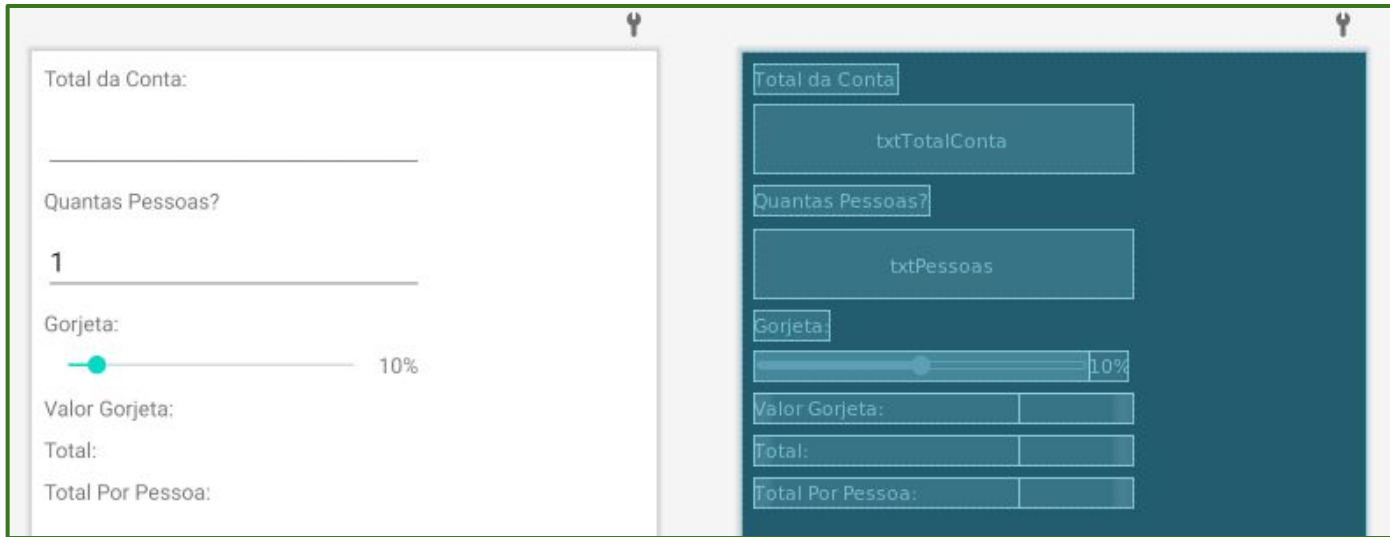
A instrução **continue** permite que o controle vá diretamente para a condição de teste e, em seguida, continue o processo de loop.

No caso do **for** a parte do incremento do loop continua.

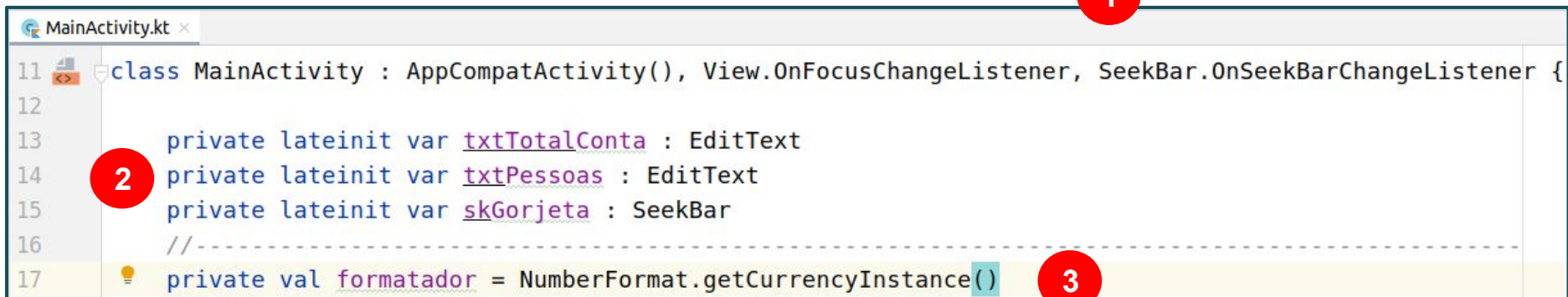
Um bom uso de uma instrução **continue** é quando você deseja reiniciar ou pular uma sequência de instrução quando ocorre um erro.

Laboratório

Calculadora de Gorjeta



Calculadora de Gorjeta



The screenshot shows the MainActivity.kt file in an IDE. Three red circles with white numbers are used as annotations: circle 1 is above the class declaration line, circle 2 is to the left of the variable declarations, and circle 3 is above the NumberFormat line.

```
11 class MainActivity : AppCompatActivity(), View.OnFocusChangeListener, SeekBar.OnSeekBarChangeListener {  
12  
13     private lateinit var txtTotalConta : EditText  
14     private lateinit var txtPessoas : EditText  
15     private lateinit var skGorjeta : SeekBar  
16     //-----  
17     private val formatador = NumberFormat.getCurrencyInstance()
```

Calculadora de Gorjeta

```
MainActivity.kt x
55  override fun onFocusChange(p0: View?, p1: Boolean) {
56      1      TODO( reason: "Not yet implemented")
57  }
58
59  override fun onProgressChanged(p0: SeekBar?, p1: Int, p2: Boolean) {
60      TODO( reason: "Not yet implemented")
61  }
62
63  override fun onStartTrackingTouch(p0: SeekBar?) {
64      2      TODO( reason: "Not yet implemented")
65  }
66
67  override fun onStopTrackingTouch(p0: SeekBar?) {
68      TODO( reason: "Not yet implemented")
69  }
70  }
```

Calculadora de Gorjeta

```
MainActivity.kt x
22  override fun onCreate(savedInstanceState: Bundle?) {
23      super.onCreate(savedInstanceState)
24      setContentView(R.layout.activity_main)
25      //-----
26      txtTotalConta = this.findViewById<EditText>(R.id.txtTotalConta)
27      txtTotalConta.setOnFocusChangeListener(this)
28      //-----
29      txtPessoas = this.findViewById<EditText>(R.id.txtPessoas)
30      txtPessoas.setOnFocusChangeListener(this)
31      //-----
32      skGorjeta = this.findViewById<SeekBar>(R.id.skGorjeta)
33      skGorjeta.setOnSeekBarChangeListener(this)
34      //-----
35  }
```

Calculadora de Gorjeta

```
MainActivity.kt
37 private fun atualizaDadosConta() {
38
39     1 if (txtTotalConta.text.toString().isNotEmpty()
40         && txtPessoas.text.toString().isNotEmpty()) {
41
42         var valorConta = txtTotalConta.text.toString().toDouble()
43         var qtdPessoas = txtPessoas.text.toString().toInt()
44
45         2 //-----
46         var lblValorRealGorjeta = this.findViewById<TextView>(R.id.lblValorRealGorjeta)
47         var valorRealGorjeta = valorConta * skGorjeta.progress / 100
48         lblValorRealGorjeta.setText(formatador.format(valorRealGorjeta))
49         //-----
50         var lblValorRealConta = this.findViewById<TextView>(R.id.lblValorRealConta)
51         lblValorRealConta.setText(formatador.format(number: valorConta + valorRealGorjeta))
52         3 var lblValorRealPorPessoa = this.findViewById<TextView>(R.id.lblValorRealPorPessoa)
53         lblValorRealPorPessoa.setText(formatador.format(number: (valorConta + valorRealGorjeta) / qtdPessoas))
54     }
```

Calculadora de Gorjeta

```
MainActivity.kt x
57  override fun onFocusChange(p0: View?, p1: Boolean) {
58      1      this.atualizaDadosConta()
59  }
60
61  override fun onProgressChanged(p0: SeekBar?, p1: Int, p2: Boolean) {
62
63      2      var lblPercentualGorjeta = this.findViewById<TextView>(R.id.lblPercentualGorjeta)
64      lblPercentualGorjeta.setText(skGorjeta.progress.toString() + "%")
65      //-----
66      this.atualizaDadosConta()
67  }
68
69  override fun onStartTrackingTouch(p0: SeekBar?) {
70  }
71
72      3  override fun onStopTrackingTouch(p0: SeekBar?) {
73  }
74  }
```




Exercício

1. Fazer funcionar as suas calculadoras do Copacabana Runners.
2. Implementar a Calculadora de Gorjeta apresentada nesta aula.