Sumário

[Conversões 1](#_Toc77858209)

[Condicionais 2](#_Toc77858210)

[Referencias 4](#_Toc77858211)

# Conversões

No Kotlin, um valor numérico de um tipo não é convertido automaticamente em outro tipo, mesmo quando o outro tipo é maior. Isso é diferente de como o Java trata as conversões numéricas. Por exemplo;

Em Java,

int number1 = 55;

long number2 = number1; // Valid code

Aqui, o valor de number1 do tipo int é automaticamente convertido para o tipo long e atribuído à variável number2.

Em Kotlin,

val number1: Int = 55

val number2: Long = number1 // Error: type mismatch.

Embora o tamanho de Long seja maior do que Int, o Kotlin não converte automaticamente Int em Long. Em vez disso, você precisa usar toLong () explicitamente (para converter para o tipo Long). Kotlin faz isso por segurança de tipo para evitar surpresas.

val number1: Int = 55

val number2: Long = number1.toLong()

Os tipos de conversão em kotlin são:

[toByte()](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.text/to-byte.html)

[toShort()](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-int/to-short.html)

[toInt()](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.text/to-int.html)

[toLong()](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.text/to-long.html)

[toFloat()](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.text/to-float.html)

[toDouble()](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.text/to-double.html)

[toChar()](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-int/to-char.html)

Observe que não há conversão para tipos booleanos.

Conversão do tipo maior para o menor

As funções mencionadas acima podem ser usadas em ambas as direções (conversão do tipo maior para o menor e conversão do tipo menor para o maior).

No entanto, a conversão do tipo maior para o menor pode truncar o valor. Por exemplo,

fun main(args : Array<String>) {

val number1: Int = 545344

val number2: Byte = number1.toByte()

println("number1 = $number1")

println("number2 = $number2")

}

saida

number1 = 545344

number2 = 64

## Condicionais

## If-else

O Kotlin apresenta vários mecanismos para implementar a lógica condicional.

O mais comum deles é uma instrução if-else. Se uma expressão entre parênteses ao lado de uma palavra-chave if for avaliada como true, o código dentro dessa ramificação (ou seja, o código imediatamente seguinte que é encapsulado entre chaves) será executado. Caso contrário, será executado o código dentro da ramificação else.

**if (count == 42) {  
    println("I have the answer.")  
} else {  
    println("The answer eludes me.")  
}**

Você pode representar várias condições usando else if. Isso permite representar uma lógica mais granular e complexa em uma única instrução condicional, conforme mostrado neste exemplo:

**if (count == 42) {  
    println("I have the answer.")  
} else if (count > 35) {  
    println("The answer is close.")  
} else {  
    println("The answer eludes me.")  
}**

As instruções condicionais são úteis para representar a lógica com estado, mas você pode se repetir ao gravá-las. No exemplo acima, você simplesmente imprime um String em cada ramificação. Para evitar essa repetição, o Kotlin oferece expressões condicionais. O último exemplo pode ser regravado da seguinte forma:

**val answerString: String = if (count == 42) {  
    "I have the answer."  
} else if (count > 35) {  
    "The answer is close."  
} else {  
    "The answer eludes me."  
}  
println(answerString)**

Implicitamente, cada ramificação condicional retorna o resultado da expressão na linha final, de modo que não é necessário usar uma palavra-chave return. Como o resultado das três ramificações é do tipo String, o resultado da expressão if-else também é do tipo String. Neste exemplo, answerString recebe um valor inicial do resultado da expressão if-else.

A inferência de tipos pode ser usada para omitir a declaração de tipo explícito para answerString, mas geralmente é uma boa ideia incluí-la para fins de clareza.

**Observação:** o Kotlin não inclui um [operador ternário](https://en.wikipedia.org/wiki/%3F:) tradicional, favorecendo o uso de expressões condicionais.

Conforme a complexidade da instrução condicional aumenta, é recomendável substituir a expressão if-else por uma expressão when, conforme mostrado neste exemplo:

**val answerString = when {  
    count == 42 -> "I have the answer."  
    count > 35 -> "The answer is close."  
    else -> "The answer eludes me."  
}**  
  
println(answerString)

Cada ramificação em uma expressão when é representada por uma condição, uma seta (->) e um resultado. Se a condição no lado esquerdo da seta for avaliada como verdadeira, o resultado da expressão no lado direito será retornado. Observe que a execução não passa de uma ramificação para a próxima. O código no exemplo de expressão when é funcionalmente equivalente ao do exemplo anterior, mas é mais fácil de ler.

As condicionais do Kotlin destacam um dos recursos mais avançados, a transmissão inteligente. Em vez de usar o operador de chamada segura ou o operador de declaração não nulo para trabalhar com valores anuláveis, você pode verificar se uma variável contém uma referência a um valor nulo usando uma instrução condicional, conforme mostrado neste exemplo:

val languageName: String? = null  
if (languageName != null) {  
    // No need to write languageName?.toUpperCase()  
    println(languageName.toUpperCase())  
}

Na ramificação condicional, languageName pode ser tratado como não anulável. O Kotlin é inteligente o suficiente para reconhecer que a condição para executar a ramificação é que languageName não contenha um valor nulo. Portanto, você não precisa tratar languageName como anulável nessa ramificação. Essa transmissão inteligente funciona para verificações nulas, [verificações de tipo](https://kotlinlang.org/docs/reference/typecasts.html#is-and-is-operators) ou qualquer condição que satisfaça a um [contrato](https://kotlinlang.org/docs/reference/whatsnew13.html#contracts).

# Referencias

<https://developer.android.com/kotlin/learn?hl=pt-br>

<https://www.w3schools.com/kotlin/index.php>