

Desenvolvimento de Aplicativos

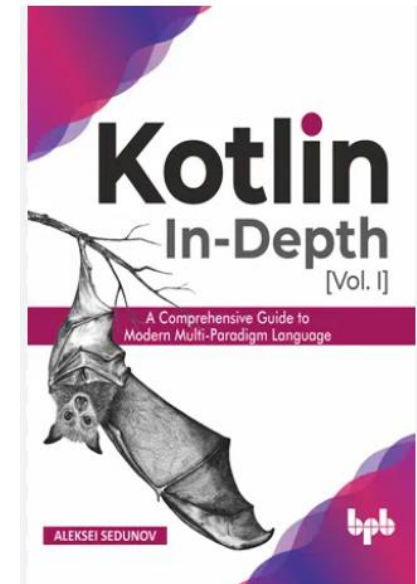
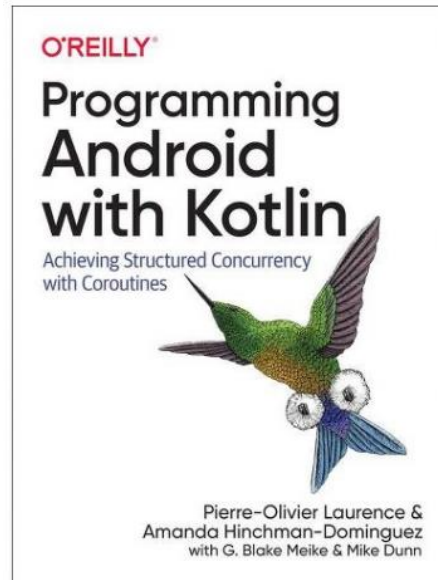
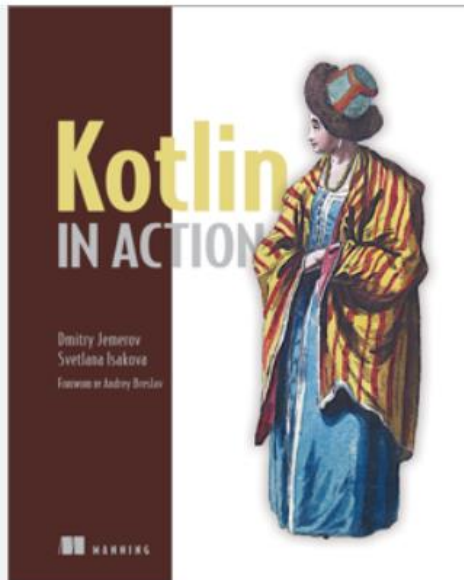
Dispositivos Móveis – Programação de
Aplicações para Clientes Móveis

Unidade 3 – Kotlin Controle de Fluxo



Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUVSP
aparecido.freitas@online.uscs.edu.br
aparecidovfreitas@gmail.com

Bibliografia



Controle de Fluxo – if

Em Kotlin, o comando `if` é usado para controle de fluxo e pode ser utilizado tanto como uma instrução condicional quanto como uma expressão.

Como uma Instrução Condicional

Como em outras linguagens de programação, `if` em Kotlin testa uma condição e executa um bloco de código se essa condição for verdadeira.

```
if (condicao) {  
    // Bloco de código que é executado se condicao é verdadeira  
}
```

Controle de Fluxo – if

Você também pode adicionar um bloco `else` para tratar o caso em que a condição é falsa.

```
if (condicao) {  
    // Bloco de código que é executado se condicao é verdadeira  
} else {  
    // Bloco de código que é executado se condicao é falsa  
}
```

if como expressão

Como uma Expressão

Uma característica única de Kotlin é que ``if`` pode ser usado como uma expressão que retorna um valor. Isso significa que você pode usar ``if-else`` para atribuir diretamente o resultado a uma variável ou retorná-lo de uma função.

```
val resultado = if (condicao) {  
    "A condição foi cumprida"  
} else {  
    "A condição não foi cumprida"  
}
```

if – pontos importantes

- A condição dentro de um `if` deve ser uma expressão booleana (`true` ou `false`).
- Como expressão, `if` deve sempre ter um ramo `else` para garantir que um valor seja retornado em todos os casos.

Usar `if` como uma expressão é uma prática comum em Kotlin e ajuda a escrever um código mais conciso e expressivo.

Quiz 3_01

O que será exibido na console?

Exercicio_3_1.kt

```
1 fun f1 (p: Double, isMember: Boolean): Double {
2     val d = if (isMember) {
3         if (p > 100) {
4             0.20
5         } else {
6             0.10
7         }
8     } else {
9         if (p > 100) {
10            0.05
11        } else {
12            0.0
13        }
14    }
15    return p * (1 - d)
16 }
```

```
fun main() {
    val p = 150.0
    val isMember = false
    println(f1(p, isMember))
}
```

- A) 120.0
- B) 135.0
- C) 150.0
- D) 140.0
- E) 142.5

Quiz 3_02

O que será exibido na console?

Exercicio_3_2.kt

```
1  //-----
2  fun calc(w: Double, isExpress: Boolean): Double {
3      val cost = if (isExpress) {
4          if (w <= 5) {
5              50.0
6          } else {
7              100.0
8          }
9      } else {
10         if (w <= 5) {
11             20.0
12         } else {
13             40.0
14         }
15     }
16     return cost
17 }
```

```
//-----
fun main() {
    val w = 7.0
    val isExpress = true
    println(calc(w, isExpress))
}
//-----
```

- A) 20.0
- B) 40.0
- C) 50.0
- D) 100.0
- E) 70.0

Comando when

- ⊕ Kotlin não tem o comando **switch**, mas tem o construto **when**.

Em Kotlin, o comando `when` é uma forma avançada e flexível de controle de fluxo, similar ao `switch` em outras linguagens como Java ou C#, mas com capacidades mais poderosas. `When` avalia uma expressão e executa o bloco de código correspondente ao primeiro caso que corresponde ao valor da expressão.

Estrutura Básica

A estrutura básica de um `when` é a seguinte:

```
when (expressao) {  
    valor1 -> {  
        // Bloco de código para valor1  
    }  
    valor2 -> {  
        // Bloco de código para valor2  
    }  
    else -> {  
        // Bloco de código se nenhum dos casos anteriores corresponder  
    }  
}
```

Quiz 3_03

O que será exibido na console?

Exercicio_3_3.kt

```
1  //-----
2  fun gResp(w: Int): String {
3      return when (w) {
4          1 -> "A"
5          2 -> "B"
6          3 -> "K"
7          else -> "Oi"
8      }
9  }
10 //-----
11 fun main() {
12     val list = arrayOf(1, 2, 3, 4)
13     var i = 0
14
15     while (i < list.size) {
16         val opt = list[i]
17         print(gResp(opt) + " ")
18         i++
19     }
20 }
```

- A) A B
- B) A B K Oi
- C) A B K
- D) K Oi
- E) Oi K B A

Comando when

Usando `when` com Valores Constantes:

```
val x = 2
when (x) {
    1 -> println("x é 1")
    2 -> println("x é 2")
    else -> println("x não é 1 nem 2")
}
```

Quiz 3_04

O que será exibido na console?

Exercicio_3_4.kt

```
1  //-----
2  fun gResp(w: Int) {
3      return when (w) {
4          1 -> print("SCS ")
5          2 -> print("SBC ")
6          3 -> print("SA ")
7          else -> print("E ")
8      }
9  }
10 //-----
11 fun main() {
12     val list = arrayOf(1, 2, 3, 4)
13     var i = 0
14
15     while (i < list.size) {
16         if (i % 2 == 0) {
17             val opt = list[i]
18             gResp(opt)
19         }
20         i++
21     }
22 }
23 //-----
```

- A) E SCS
- B) SCS SA
- C) SA E
- D) E SA
- E) SBC SA

Comando when

Vários Valores em um Único Caso:

```
val x = 2
when (x) {
    0, 1 -> println("x é 0 ou 1")
    else -> println("x é maior que 1")
}
```

Quiz 3_05

O que será exibido na console?

Exercicio_3_5.kt

```
1  //-----
2  fun gResp(w: Int) : String {
3      return when (w) {
4          0,1      -> "A"
5          4,5,6    -> "B"
6          else     -> "X"
7      }
8  }
9  //-----
10 fun main() {
11     val list = arrayOf(1, 2, 3, 4, 5, 6, 7)
12     var i = 0
13
14     while (i < list.size) {
15         if (i % 2 == 0) {
16             val opt = list[i]
17             print (gResp(opt) + " ")
18         }
19         i++
20     }
21 }
```

- A) A X A X
- B) B X A X
- C) A X B X
- D) X A X A
- E) A B A B

Comando when

Usando Intervalos em `when`:

```
val x = 5
when (x) {
    in 1..4 -> println("x está entre 1 e 4")
    in 5..10 -> println("x está entre 5 e 10")
    else -> println("x está fora do intervalo 1-10")
}
```

Quiz 3_06

O que será exibido na console?

```
//-----  
fun main() {  
    val list = arrayOf(1, 2, 3, 4, 5, 6, 7)  
    var i = 0  
  
    while (i < list.size) {  
        if (i % 2 == 1) {  
            val opt = list[i]  
            print (gResp(opt) + " ")  
        }  
        i++  
    }  
}
```

```
//-----  
fun gResp(w: Int) : String {  
    return when (w) {  
        in 1..4 -> "W"  
        in 0..8 -> "R"  
        else    -> "U"  
    }  
}
```

- A) W R W R
- B) W W R R
- C) R W R W
- D) W W R R
- E) U R U W

Comando when

`when` como Expressão:

```
val x = 2
val str = when (x) {
    1 -> "x é 1"
    2 -> "x é 2"
    else -> "x é outro número"
}
println(str)
```

Quiz 3_07

O que será exibido na console?

Exercicio_3_7.kt

```
1 fun main() {
2     val list = arrayOf(1, 2, 3, 4, 5, 6, 7)
3     var i = 0 ; var w = 0
4
5     while (i < list.size) {
6         if (i % 2 == 1)
7             w = 1
8         else
9             w = 4
10        i++
11    }
12    print ( when (w) {
13        1 -> "um"
14        2 -> "dois"
15        3 -> "três"
16        4 -> "quatro"
17        else -> "outro"
18    } )
19 }
```

- A) um
- B) dois
- C) três
- D) quatro
- E) outro

Quiz 3_07

O que será exibido na console?

Exercicio_3_7.kt

```
1 fun main() {  
2     val list = arrayOf(1, 2, 3, 4, 5, 6, 7)  
3     var i = 0 ; var w = 0  
4  
5     while (i < list.size) {  
6         if (i % 2 == 1)  
7             w = 1  
8         else  
9             w = 4  
10        i++  
11    }  
12    print ( when (w) {  
13        1 -> "um"  
14        2 -> "dois"  
15        3 -> "três"  
16        4 -> "quatro"  
17        else -> "outro"  
18    } )  
19 }
```

- A) um
- B) dois
- C) três
- D) quatro
- E) outro

Comando while

A estrutura `while` executa o bloco de código enquanto a condição especificada é verdadeira. A condição é verificada antes da execução do bloco de código em cada iteração.

```
while (condicao) {  
    // Bloco de código a ser repetido  
}
```

Exemplo:

```
var contador = 0  
while (contador < 5) {  
    println("Contador é $contador")  
    contador++  
}
```

Comando do while

O comando `do-while` é semelhante ao `while`, mas a diferença principal é que no `do-while` o bloco de código é executado pelo menos uma vez, pois a condição é verificada após a execução do bloco.

```
do {  
    // Bloco de código a ser repetido  
} while (condicao)
```

Comando do while

```
var contador = 0
do {
    println("Contador é $contador")
    contador++
} while (contador < 5)
```

Neste exemplo, o bloco de código dentro do `do` será executado primeiro e, em seguida, a condição `contador < 5` será verificada. O bloco será repetido enquanto a condição for verdadeira.

Quiz 3_08

O que será exibido na console?

```
Exercicio_3_8.kt
1  fun main() {
2
3      var x = 0;
4      var y = 5;
5
6      while (x < 5) {
7          y += x;
8          x++;
9      }
10
11     do {
12         y -= 1;
13     } while (y > 10)
14
15     val r: Int = x + y;
16     println("$r");
17 }
```

- A) 13
- B) 14
- C) 15
- D) 16
- E) 17

Comando for

Em Kotlin, o comando `for` é usado para iterar sobre uma coleção ou intervalo de elementos. Ele é frequentemente utilizado para percorrer arrays, listas, conjuntos, intervalos e outras estruturas que implementam a interface `Iterable`.

Estrutura Básica do Comando `for`

```
for (item in colecao) {  
    // Bloco de código a ser executado para cada elemento  
}
```

Aqui, `item` é uma variável que representa o elemento atual na coleção ou intervalo `colecao` a cada iteração do loop.

Comando for

```
for (i in 0 until 5) {  
    println(i)  
}
```

```
0  
1  
2  
3  
4
```

Comando for

Iterando sobre um Array ou Lista:

```
val numeros = arrayOf(1, 2, 3, 4, 5)
for (numero in numeros) {
    println(numero)
}
```

Este exemplo percorre um array de inteiros, imprimindo cada número.

Iterando sobre um Intervalo:

```
for (i in 1..5) {
    println(i)
}
```

Este exemplo itera sobre um intervalo de 1 a 5.

Comando for

Iterando em Ordem Decrescente:

```
for (i in 5 downTo 1) {  
    println(i)  
}
```

Itera de 5 a 1 em ordem decrescente.

Iterando com Passos:

```
for (i in 1..10 step 2) {  
    println(i)  
}
```

Itera de 1 a 10 com um passo de 2 (ou seja, imprime números ímpares).

Comando for

```
val fruits = arrayOf("Apple", "Banana", "Cherry")  
for (fruit in fruits) {  
    println(fruit)  
}
```

- Resultado:

```
Apple  
Banana  
Cherry
```

Comando for

```
val text = "Kotlin"  
for (char in text) {  
    println(char)  
}
```

- Resultado:

```
K  
o  
t  
l  
i  
n
```

Quiz 3_09

O que será exibido na console?

Exercicio_3_9.kt

```
1 fun main() {  
2     val lista = arrayOf(1, 2, 3, 4, 5);  
3     var r = 0;  
4  
5     for (n in lista) {  
6         if (n % 2 == 0) {  
7             r += n;  
8         }  
9     }  
10  
11     for (i in 1..5 step 2) {  
12         r *= i  
13     }  
14  
15     println(r)  
16 }
```

- A) 88
- B) 90
- C) 92
- D) 94
- E) 96

Quiz 3_010

O que será exibido na console?

Exercicio_3_10.kt

```
1 fun main() {  
2     val lista = arrayOf(1, 2, 3, 4)  
3     var r = 1  
4  
5     for (n in lista) {  
6         if (n % 2 != 0) {  
7             continue  
8         }  
9         r += n  
10    }  
11  
12    for (i in 1..r step 2) {  
13        if (i % 4 == 0) {  
14            continue  
15        }  
16        r += i  
17    }  
18  
19    println(r)  
20 }
```

A) 20

B) 21

C) 22

D) 23

E) 24

Entrada de Dados pela Console

Em Kotlin, para ler entrada de dados do console, você pode usar a função `readLine()`. Esta função é usada para ler uma linha de entrada do console como uma string. Se você precisar de tipos de dados específicos além de strings, como `Int`, `Double`, etc., você pode converter a string lida para o tipo de dado desejado.

A função `readLine()` é a forma mais direta e comumente usada para leitura de entrada do console em Kotlin, funcionando de maneira similar ao `Scanner` em Java, mas de forma mais concisa para algumas operações básicas.

Entrada de Dados pela Console

Se você está enfrentando dificuldades para realizar execução interativa de programas Kotlin no VSCode, especialmente com funções como `readLine()` para entrada de dados, aqui estão algumas etapas que você pode seguir para tentar resolver o problema:

1. Verifique a Configuração do Code Runner

Se você estiver usando a extensão Code Runner para executar seus scripts Kotlin, certifique-se de que a execução está configurada para ocorrer no terminal integrado, o que permite interações com o usuário:

1. Abra as configurações do VSCode (`Ctrl` + `,` ou Cmd` + `,` no macOS).`
2. Procure por `Code Runner: Execute in Terminal`.`
3. Verifique se esta opção está ativada. Isso assegura que os scripts serão executados no terminal integrado, permitindo entrada de dados pelo usuário.

Entrada de Dados pela Console

Como `readLine()` Funciona?

- Quando `readLine()` é chamada, o programa aguarda até que o usuário digite algo e pressione Enter. Em seguida, o que foi digitado é capturado como uma `String`.
- Se a função `readLine()` for capaz de capturar a entrada, ela retornará essa entrada como uma `String`.
- Se houver alguma condição excepcional (como o fim do fluxo de entrada), `readLine()` pode retornar `null`.

Entrada de Dados pela Console

Capturando Entradas simples

- Exemplo: Capturando o nome do usuário

kotlin

```
fun main() {  
    println("Digite seu nome:")  
    val nome = readLine()  
    println("Olá, $nome!")  
}
```

- Explicação:
 - ``println("Digite seu nome:")``: Solicita ao usuário que digite o nome.
 - ``val nome = readLine()``: Captura a entrada do usuário.
 - ``println("Olá, $nome!")``: Exibe uma saudação usando o nome digitado.

Entrada de Dados pela Console

Convertendo entradas numéricas

Exercicio_3_11.kt

```
1  fun main() {
2      println("Entre com um texto:")
3      val texto = readLine();
4
5      println("Entre com um valor numérico:")
6      val n: Int?
7      n = readLine()?.toIntOrNull();
8
9      if (n != null) {
10         println("O número digitado foi: $n")
11
12         for (i in 1..n) {
13             println("$i: $texto")
14         }
15     } else {
16         println("Entrada inválida!")
17     }
18 }
```

Entrada de Dados pela Console

Convertendo entradas numéricas

```
val numero: Int? = 42 // 'numero' pode ser 42  
val outroNumero: Int? = null // 'outroNumero' pode ser null
```

- O que é? **Int?**
- indica que a variável **numero** pode armazenar um valor do tipo **Int** (um número inteiro) ou pode ser **null**.
- A interrogação no final de **Int** significa que a variável pode ser nula (**null**);
- A interrogação ao final do **Int** é chamada indicador de nulabilidade.

```
val numero: Int = 42  
numero = null // ERRO: Isso não é permitido
```

Indicador de nulabilidade

Tipos Nulos (`Int?`):

- Quando você declara uma variável como `Int?`, você está dizendo que essa variável pode conter um número inteiro ou pode ser `null`.
- Exemplo:

```
val numero: Int? = 42  
numero = null // Isso é permitido
```

Por que Isso é Importante?

Kotlin foi projetado para ser uma linguagem segura em relação a nulidade (null safety). Muitos erros em linguagens como Java ocorrem devido a `NullPointerException`, que acontece quando você tenta usar um objeto que é `null`. Kotlin minimiza esses erros forçando o programador a lidar com `null` de forma explícita.

Entrada de Dados pela Console

Porque isso é útil ?

- Em Kotlin, diferenciar entre uma variável que deve ter um valor (`Int`) e uma que pode ser nula (`Int?`) é importante para evitar erros.
- Por exemplo, se uma variável for **null** e você tentar usá-la como se tivesse um valor, pode causar problemas no seu programa (como exceções de **NullPointerException**).

```
val n: Int?  
n = readLine()?.toIntOrNull();  
  
if (n != null) {  
    println("O número digitado foi: $n")  
}  
else {  
    println("Entrada inválida!")  
}
```


Entrada de Dados pela Console

Conversão de Tipos

Note que usamos o operador de chamada segura (`?.`) junto com `toInt()` para converter a string lida para um inteiro. Isso é porque `readLine()` pode retornar `null` se nenhuma linha for lida (por exemplo, se o final da entrada for alcançado), e `toInt()` é chamado apenas se `readLine()` não retornar `null`. Se a conversão não for possível (por exemplo, se o usuário digitar texto que não pode ser convertido para um número), o programa lançará uma exceção. Para lidar com isso de forma mais robusta, você pode usar `toIntOrNull()` e verificar se o resultado é `null`:

Entrada de Dados pela Console

kotlin

 Copy code

```
val idadeString = readLine()
val idade = idadeString?.toIntOrNull()

if (idade != null) {
    println("Você tem $idade anos.")
} else {
    println("Idade inválida.")
}
```

Quiz 3_11

O que será exibido na console, considerando que o usuário digitou 3 e 8 ?

Exercicio_3_11.kt

```
1  fun main() {  
2  
3      var r = 0;  
4      val a = readLine()?.toIntOrNull();  
5      val b = readLine()?.toIntOrNull();  
6  
7      if (a != null && b != null) {  
8          for (i in a..b) {  
9              r += (i + a%i + b%i);  
10         }  
11     }  
12     println(r);  
13 }
```

A) 53

B) 54

C) 55

D) 56

E) 57

Quiz 3_12

O que será exibido na console, considerando que o usuário digitou 4 e 7 ?

Exercicio_3_12.kt

```
1 fun main() {  
2  
3     var r = 0  
4     val a = readLine()?.toIntOrNull()  
5     val b = readLine()?.toIntOrNull()  
6  
7     if (a != null && b != null && a < b) {  
8         for (i in a..b) {  
9             if (i % 2 == 0) {  
10                r += (i * 2 + a / (i + 1) + b / (i + 2))  
11            } else {  
12                r += (i + a % (i + 1) + b % (i + 2))  
13            }  
14        }  
15    }  
16    println(r)  
17 }
```

A) 47

B) 48

C) 49

D) 50

E) 51

Exception Handling

Em Kotlin, o tratamento de exceções (Exception Handling) é um mecanismo para lidar com erros de tempo de execução de uma maneira controlada. Ele permite que você defina blocos de código que são executados quando ocorrem erros (exceções) e permite que você recupere o fluxo normal do programa. O tratamento de exceções em Kotlin é muito similar ao de Java, dado que ambos usam o modelo de exceções verificadas e não verificadas.

Estrutura Básica do Tratamento de Exceções


O tratamento de exceções em Kotlin é feito principalmente por meio dos comandos ``try``, ``catch`` e ``finally``.

Exception Handling

1. Bloco `try`:

- O bloco `try` contém o código que pode gerar uma exceção. Se uma exceção ocorrer dentro deste bloco, ela é jogada, e o fluxo de execução é transferido para o bloco `catch` correspondente.

kotlin

 Copy code


```
try {  
    // Código que pode gerar uma exceção  
}
```

Exception Handling

2. Bloco `catch`:

- O bloco `catch` é usado para capturar a exceção lançada pelo bloco `try`. É aqui que você pode lidar com a exceção, seja registrando um erro, tomando ações corretivas, etc.

kotlin

 Copy code


```
catch (e: SomeException) {  
    // Código para lidar com a exceção  
}
```

Exception Handling

3. Bloco `finally` (opcional):

- O bloco `finally` é opcional e é executado independentemente de uma exceção ter sido lançada ou capturada. É geralmente usado para limpar recursos, como fechar arquivos ou conexões de banco de dados.

kotlin

 Copy code

```
finally {  
    // Código que sempre é executado após os blocos try/catch  
}
```

Exception Handling

Exemplo de Tratamento de Exceções

kotlin

```
try {  
    val resultado = someFunctionThatMightThrowException()  
    println(resultado)  
} catch (e: SomeSpecificException) {  
    println("Ocorreu uma exceção específica: ${e.message}")  
} catch (e: Exception) {  
    println("Ocorreu uma exceção genérica: ${e.message}")  
} finally {  
    println("Este bloco é sempre executado.")  
}
```