Aula 12

Estrelas fora da caixa

ZUP

Kotlin

O que é o Kotlin?

"Linguagem de programação moderna, concisa e segura
Fácil de pegar, para que você possa criar aplicativos poderosos imediatamente."

Onde usar?

- Multiplatform Mobile
- Server-side
- Web front-end
- Android

Sintaxe básica

- Definição de pacotes e imports
- Ponto de entrada do programa ou Ponto de execução
- Saída padrão de dados
- Entrada padrão de dados
- Tipo de dados
- Variáveis
- Conversões explícitas
- String templates
- Tipos anuláveis e tipos não nulos
- Estrutura condicional

Definição de pacotes e imports

```
package my.demo

import kotlin.text.*

// ...
```

Detalhe importante:

Não precisamos usar o **;** no Kotlin

Ponto de entrada ou Ponto de execução

```
fun main() {
    println("Hello world!")
}
```

Ponto de entrada ou Ponto de execução

```
fun main(args: Array<String>) {
    println(args.contentToString())
}
```

- VARIÁVEIS

Variáveis - val

As variáveis de leitura podem recer um valor apenas uma vez e são definidas usando a palavra-chave **val**

```
fun main() {
    val a: Int = 1  // atribuição imediata
    val b = 2   // O tipo `Int` é inferido
    val c: Int  // Tipo necessário quando nenhum inicializador é fornecido
    c = 3       // atribuição adiada
}
```

Variáveis - var

Variáveis que podem ser reatribuídas usam a palavra-chave var

```
fun main() {
    var x = 5 // O tipo `Int` é inferido
    x += 1
}
```

- SAÍDA E ENTRADA DE DADOS

Saída padrão de dados

```
fun main() {
    print("Mensagem na mesma")
    print("linha")

    println("Mensagem que pula uma linha")
}
```

Entrada padrão de dados - readln()

```
fun main() {
    println("Digite alguma informação importante: ")
    val informacao = readln()
    println(informacao)
}
```

Entrada padrão de dados - readln() e conversões

```
fun main() {
    println("Digite um texto: ")
    val texto = readln()

    println("Digite um numero inteiro: ")
    val numero = readln().toInt()

    println("Digite um numero double: ")
    val numeroDouble = readln().toDouble()

    println("Digite um float: ")
    val numeroFloat = readln().toFloat()

    println("Digite um boolean: ")
    val valorBoolean = readln().toBoolean()
}
```

- TIPOS DE DADOS
- CONVERSÕES EXPLÍCITAS
- STRING TEMPLATES

Tipos de dados - Inteiros

Type	Size (bits)	Min value	Max value
Byte	8	-128	127
Short	16	-32768	32767
Int	32	-2,147,483,648 (-2^{31})	2,147,483,647 ($2^{31} - 1$)
Long	64	-9,223,372,036,854,775,808 (-2^{63})	9,223,372,036,854,775,807 ($2^{63} - 1$)

Tipos de dados - Exemplo Inteiros

```
fun main() {
    val um = 1 // Int
    val tresBilhoes = 3000000000 // Long
    val umLong = 1L // Long
    val umByte: Byte = 1 //Byte
}
```

Tipos de dados - Ponto flutuante

Type	Size (bits)	Min value	Max value
Float	32	24	8
Double	64	53	11

Tipos de dados - Exemplo Ponto flutuante

```
fun main() {
    val pi = 3.14 // Double
    val umDouble = 1.0 // Double

    val e = 2.7182818284 // Double
    val eFloat = 2.7182818284f // Float, valor atual é 2.7182817
}
```

Tipos de dados - Booleans

|| – disjunção (OU)
&& – conjunção (E)
! - negação (NÃO)

Tipos de dados - Booleans

```
fun main() {
    val myTrue: Boolean = true
    val myFalse: Boolean = false
    val boolNull: Boolean? = null

    println(myTrue || myFalse)
    println(myTrue && myFalse)
    println(!myTrue)
}
```

Tipos de dados - Characters

Os caracteres são representados pelo tipo Char. Os literais de caracteres vão entre aspas simples: '1'.

Caracteres especiais começam com uma barra invertida \. As seguintes sequências de escape são suportadas: \t, \b, \n, \r, \', \", \\ e \$.

Para codificar qualquer outro caractere, use a sintaxe de sequência de escape Unicode: '\uFF00'.

Tipos de dados - Exemplo Characters

```
fun main() {
    val aChar = 'a'

    println(aChar)
    println('\n') //prints an extra newline character
    println('\uFF00')
}
```

Tipos de dados - Strings

```
fun main() {
    val str = "abcd"
    println(str.uppercase()) // Cria e mostra um texto em uppercase
    println(str) // mostra o texto original
}
```

Conversões explícitas

Todos os tipos de números são compatíveis com conversões para outros tipos:

- toByte(): Byte
- toShort(): Short
- toInt(): Int
- toLong(): Long
- toFloat(): Float
- toDouble(): Double
- toChar(): Char

String templates

Concatenando valores com String templates

```
fun main() {
    val i = 10
    println("i = $i") // exibe "i = 10"

    val s = "abc"
    println("$s.length is ${s.length}") // exibe "abc.length is 3"
}
```

- TIPOS ANULÁVEIS E NÃO NULOS

Tipos anuláveis e tipos não nulos

- O sistema de tipos do Kotlin visa eliminar o perigo de referências nulas.
- NullPointerException
- As únicas causas possíveis de um NPE em Kotlin são:
  - Uma chamada explícita para lançar NullPointerException()
  - Uso do !! operador descrito abaixo.

Tipos anuláveis e tipos não nulos - Exemplos

Exemplo de uma variável regular do tipo String que não pode conter null:

```
fun main() {
    var a: String = "abc" // Inicialização regular significa não nulo por padrão
    a = null // erro de compilação
}
```

Tipos anuláveis e tipos não nulos - Exemplos

Para permitir nulos, você pode declarar por exemplo uma variável como uma string anulável escrevendo String**?**

```
fun main() {
    var b: String? = "abc" // pode receber um valor null
    b = null // ok
    print(b)
}
```

Tipos anuláveis e tipos não nulos - Chamadas seguras

Para acessar uma propriedade em uma variável anulável é usar o operador de chamada segura **?.**

```
fun main() {
    val a = "Kotlin"
    val b: String? = null
    println(b?.length) //Chamada segura
    println(a?.length) // Desnecessário uma chamada segura
}
```

Tipos anuláveis e tipos não nulos - Elvis Operator

Quando você tem uma referência anulável, b, você pode dizer "se b não for nulo, use-o, caso contrário, use algum valor não nulo"

```
fun main() {
    val nome = "Kotlin"
    val sobrenome: String? = null
    val tamanhoSobrenome = sobrenome?.length ?: -1
}
```

Tipos anuláveis e tipos não nulos - Operador !!

O operador de afirmação não nulo (!!) converte qualquer valor em um tipo não nulo e lança uma exceção se o valor for nulo

```
fun main() {
    val nome = "Kotlin"
    val sobrenome: String? = null
    val tamanhoSobrenome = sobrenome!!.length
}
```

- CONDICIONAIS

Condicionais - if else

Exemplo 1

```
fun main() {
    val a = 2
    val b = 3
    var max = a

    if (a < b) max = b
}
```

Condicionais - if else

Exemplo 2

```
fun main() {
    val a = 2
    val b = 3

    // Como uma expressão
    val max = if (a > b) a else b
}
```

Condicionais - if else

Exemplo 3

```
fun main() {
    val a = 2
    val b = 3

    // Com else
    var max: Int
    if (a > b) {
        max = a
    } else {
        max = b
    }
}
```

Condicionais - when

when define uma expressão condicional com várias ramificações. É semelhante à instrução switch em linguagens semelhantes a C

```
fun main() {
    val x = 3
    when (x) {
        1 -> print("x == 1")
        2 -> print("x == 2")
        else -> {
            print("x is neither 1 nor 2")
        }
    }
}
```

Hora da prática...

Material de Apoio

- Basic syntax
- Conditions and loops
- Null safety