TI-2600 Laboratório de Desenvolvimento Multiplataforma

Antonio Carvalho - Treinamentos

Kotlin

O que é Kotlin?

- Uma linguagem (justamente) nova e tipada estaticamente da JetBrains (criadores do IntelliJ Idea, ReSharper, PyCharm e outros IDEs e extensões IDE)
- Tenta corrigir muitas das deficiências do Java
- Compila para bytecode JVM, JavaScript (!) e Kotlin nativo (sem VM)
- Criado com foco na interoperabilidade Java – as classes Kotlin e Java podem ser usadas juntas em um projeto (mas a compilação é muito mais rápida do que a do Scala)

O que é Kotlin?

- Funciona perfeitamente com o IntelliJ Idea, apenas um pouco menos perfeitamente com o Eclipse
- A interoperabilidade Java é real – até mesmo aplicações complexas, contando com processamento de anotações
- Tem muito bom apoio da JetBrains e da Google, uma empresa estabelecida, que usam Kotlin para desenvolver seus próprios produtos

Características da linguagem

- **Linguagem estaticamente tipada.**
- Mais concisa, reduzindo o montante de códigos necessários para criar estruturas. Por exemplo é possível criar **getters**, **setters**, **equals**(), **hashCode**(), **toString**() e **copy**() com uma única linha:
- `data class Cliente(val nome: String, val telefone: String, val email: String)`

Características da linguagem

- **É interoperável com Java**
- Ou seja é possível colocar código Java misturado com código Kotlin

```
println("Hello World")
System.out.println("Ola mundo")
```

Características da linguagem

- **Evita o NullPointerException**

```
var b : String? = "abc"
b = null // Ok
```

Características da linguagem

- **Sintaxe familiar**
- Em alguns casos a sintaxe é semelhante ao Java, a UML e ao Python

```
class Aluno : Pessoa() {
   ...
}
```

Características da linguagem

- **Interpolação de String**
- É possível colocar valores de variáveis dentro de Strings

```
val x = 4
val y = 7
print("soma de $x e $y é igual a ${x + y}")
```

Características da linguagem

- **Inferência de tipos**
- Os tipos das variáveis são determinados na atribuição de dados, porém é possível determinar o tipo com antecedência se preferir.

```
val a = "abc" // Tipo inferido como String
val c : Double = 0.7
// Tipo declarado de forma explicita
```

Características da linguagem

- **Cast (Conversão) Inteligente**
- O Kotlin rastreia a lógica e converte os tipos automaticamente quando possível.

```
if (obj is String) {
   print(obj.toUpperCase())
}
// assume que obj é uma String
```

Características da linguagem

- **Igualdade intuitiva**
- Não é mais necessário o uso do **.equals()** na comparação entre objetos porque o == já fará esta atividade de verificar a igualdade entre estruturas

```
val p1 = Pessoa("João")
val p2 = Pessoa("João")
p1 == p2 // true (Igualdade estrutural)
p1 === p2 // false (Igualdade referencial)
```

Características da linguagem

- **Argumentos padrões**
- Não é mais necessário fazer diversas sobrecargas nas funções para assumir argumentos padrões. O Kotlin aceita argumentos padrões desde o princípio

```
fun criarTela (titulo : String,
   x : Int, y : Int,
   width : Int = 800, height: Int = 600) {
   Frame(titulo, x, y, width, height)
}
```

Características da linguagem

- **Argumentos nomeados**
- Assim como no **python** e **Javascript** é possível utilizar argumentos nomeados nas chamadas das funções podendo alterar a ordem em que são passados

```
criarTela("Mario Bros", 50, 100)
criarTela(titulo="Android Teste", x=50, y=100)
criarTela(width=400, height=800,
   titulo="React Native Teste", x=50, y=100)
```

Características da linguagem

- **Classes de Dados**
- É possível criar classes de dados como se fossem classes Java normais com gets, sets, equals, copy, sem a necessidade de dezenas de linhas de código

```
data class Pessoa(val nome: String,
val email: String = "teste@teste.com",
val telefone: String = "(11) 1111-1111")

val p1 = Pessoa(nome="João")
```

Características da linguagem

- **Sobrescrita de operadores**
- Um conjunto de operadores predefinidos podem ser sobrescritos para melhorar a visibilidade

```
data class Vec(val x: Float, val y: Float) {
    operator fun plus(v: Vec) =
        Vec( x + v.x, y + v.y)
}

val v = Vec(2f, 3f) + Vec(4f, 1f)
```

Características da linguagem

- **Tipos nullables**
- O Kotlin assegura operações que evitem o NullPointerException proporcionando dois tipos de universos os elementos nuláveis e não nuláveis
- Elementos que podem aceitar valores nulos possuem um sufixo (?)

```
var texto : String?  // Aceita valores nulos
```

- Elementos que não aceitam valores nulos não possuem sufixo

```
var texto : String  // Não Aceita valores nulos
```

Características da linguagem

- **Tipos nullables**
- Para operações em ? T? que pode violar a segurança nula, por exemplo, o acesso a uma propriedade, tem-se as seguintes opções nulas-safe:
- Use operações seguras
  - *Cast* seguro
- Reduzir de ? T? Para!! T!!
  - *Cast* inseguro
  - verificação de tipo combinada com smart casts
  - verificação nula combinada com smart casts
  - operador de asserção não-nulo
- Forneça um valor padrão a ser usado se null estiver presente
  - elvis operador **?: <valor padrão em caso nulo>**

Características da linguagem

- **Funções de extensão**
- É possível melhorar as classes criando funções adicionais a elas sem a necessidade de fazer a herança

```
fun String.replaceSpaces() : String {
   val result = this.replace(' ', '_')
   return result
}

val formatada = "nome da aplicação".replaceSpaces()
```

Estrutura do código

Comentários

Pode ser utilizado da seguinte maneira:

// indica que a partir daquele ponto até o final da linha tudo será considerado como comentário

Exemplo:

```
var x = 10
x = x + 1                    // Incrementa x
```

/* indica inicio e ***/** indica o fim de comentário em bloco

Exemplo:

```
/*
    Comentário que pode ser colocado em
    várias linhas
    assim como em outras linguagens   */
```

Comentários

/** indica inicio e ***/** indica o fim de comentário em bloco para efeito de documentação, deve ser colocado após a linha de declaração de **class** ou **fun**

Exemplo:

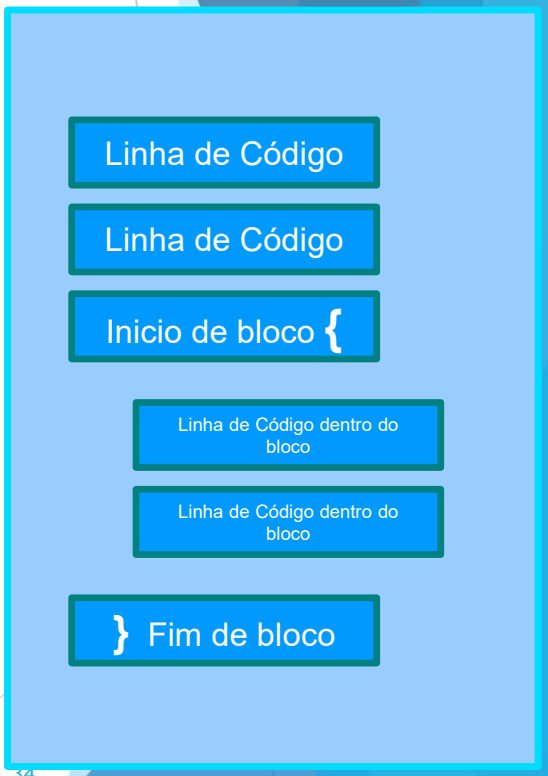
```
fun somaNumeros( n1 : Int, n2 : Int ) {
    /**
    Comentário a ser usado na documentação do código
    também com múltiplas linhas
    */
    return n1 + n2

}
```

Bloco de código

O bloco de código inicia-se com { e termina com }

Exemplo :

```
var soma = 0
for (i in 0..10) {
    println(i)
    soma = soma + 1
}
println("Soma $soma")
```



Palavras reservadas

as	break	class	continue	do
else	false	for	fun	if
in	interface	is	null	object
package	return	super	this	throw
true	try	typealias	typeof	val
var	when	while		

Gramática Completa
https://kotlinlang.org/docs/reference/grammar.html

Tipos de dados e variáveis

Tipos de dados e Variáveis

Variável primitiva	Tipo da informação	Tamanho na memória	Faixa
Boolean	Valor lógico **true** ou **false**	1 bit	false, true
Byte	Número inteiro	8 bits	$-2^7 .. 2^7 - 1$
Short	Número inteiro	16 bits	$-2^{15} .. 2^{15} - 1$
Int	Número inteiro	32 bits	$-2^{31} .. 2^{31} - 1$
Long	Número inteiro	64 bits	$-2^{63} .. 2^{63} - 1$
Float	Número decimal	32 bits	varia
Double	Número decimal	64 bits	varia
Char	Charactere (apenas um)	16 bits	$0 .. 2^{16} - 1$

Tipos de dados e Variáveis

Variável primitiva	Tipo da informação	Tamanho na memória	Faixa
UByte	Número inteiro	8 bits	$0.. 2^8 – 1$
UShort	Número inteiro	16 bits	$0 .. 2^{16} – 1$
UInt	Número inteiro	32 bits	$0 .. 2^{32} – 1$
ULong	Número inteiro	64 bits	$0 .. 2^{64} – 1$

Conversões de tipos entre números

Todos os números possuem funções para conversão

Funções para conversão:

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

Tipos de dados e Variáveis

Strings

Descrição: Classe que gera objeto imutável que possibilita conter texto. A String é iterável e seus caracteres podem ser acessados por meio de um índice entre [ ]

Métodos mais utilizados :

.length	Retorna o tamanho da String
.uppercase()	Retorna o texto da String em maiúsculo
.lowercase()	Retorna o texto da String em minúsculo
.indexOf(String sub)	Retorna a posição do texto dentro da String principal

Exercício

- Complete as linhas de modo que o resultado seja verdadeiro:

var a = 10 + ____ // 30

var b = " O valor do numero é $____ " // " O valor do número é 30 "

var c = ____.toHexString() // DB75C

var d = "Ola mundo“.length // ____

var e = "153" ____ + 160 // 313

var f = "PROVA PROGRAMAÇÃO"
f.indexOf("P", ____) # 6

Operadores

Operadores

Atribuição

=	Atribuição	`var a = 4` `var b = 5.3` `var c = "Texto"`

Aritméticos

+, -, *, /	Adição, subtração, multiplicação e divisão	`var a = 0 // 0` `var b = a + 10 / 2 // 5` `var c = b - 5 * 3 // -10`
%	Divisão resultando Resto	`var a = 11 // 11` `var b = a % 3 // 2`

Operadores

- **Relacionais**

>, >=, <, <=	Maior, maior igual, menor, menor igual	`var a = 4` `var b = 0` `var c = a > b` `println("A: $a  B:$b  C(a>b):$c”)  // A: 4  B: 0  C (a>b): true` `if (a > b) {` `  b = a` `} else {` `  if a < b {` `  a = b` `  }` `}` `println("Valor de a:$a  valor b:$b”)`
==, !=	Igual e diferente	`var a = 4` `var b = 0` `var msg = ""` `if (a == b) {` `  msg = "são iguais“` `}` `if (a != b) {` `  msg = "são diferentes“` `}` `println(" Os valores de a e b $msg")`

Operadores

- **Lógicos**

! (not), &&(and), || (or), xor	**(!) Não (and) E (or) OU (xor) OU Exclusivo**	`var a = 4` `var b = 0` `var msg = ""` `if ((a == b) or (a > 5)) {` `  msg = "1° if verdadeiro"` `}` `if ((a != b) and !(b > 10)) {` `  msg = msg +  "2° if verdadeiro"` `}` `println(msg)`

Operadores

- **Atribuição aritmética**

+=, -=, *=, /=, %=	Atribuição com operação	`var b = 5` `b += 10  // 15` `b *= 3  // 45`

- **Unário**

-	Inverte o sinal aritmético do número ou da variável numérica	`var a = 5` `b = -a  // -5`

Operadores

- **Pertencimento**

in **!in**	**Há vários usos para o operador in como exemplo:** ❖ **(in) Especifica o objeto que está sendo iterado em um loop for.** ❖ **(in e !in) É usado como um operador para verificar se um valor pertence a um intervalo, uma coleção ou outra entidade que define um método 'contains'.**	`val lista = arrayListOf(2, 4, 6, 8, 10, 12)` `for (i in 0..<lista.size) {` `println("Indice: $i  Valor: ${lista[i]}")` `}` `if (14 in lista) {` `println("Numero 14 pertence a lista")` `} else {` `println("Numero 14 não pertence a lista")` `}`

Operadores

- **Identidade**

is **!is**	**is** - retorna True se o elemento x for do mesmo tipo que o elemento y e False caso não seja **!is** - retorna False se o elemento x for do mesmo tipo que o elemento y e True caso não seja	`open class Pessoa {` `  var nome = ""` `}` `open class Aluno : Pessoa() {` `  var matricula = ""` `}` `fun main() {` `  val a1 : Pessoa = Aluno()` `  val p1 = Pessoa()` `  if (a1 is Aluno) {` `  println("Variavel p1 é do tipo Aluno")` `  } else {` `  println("Variavel p1 não é do tipo Aluno")` `  }` `}`

Operadores

- **Bits**

shl **shr**	Deslocamento de bits Esquerda Direita	`val numero = 63` `val numDeslocadoEsquerda = numero shl 2` `print("Resultado $numero << 2 => $numDeslocadoEsquerda")` `// Resultado 63 << 2 => 252`

Classes

Classes em Kotlin

- O Kotlin possui diversos tipos diferentes de classes, vamos abordar as mais simples e comuns sendo elas:
  - Classe Normal
  - Data Class
  - Enum Class
  - Abstract Class
  - Companion Object
  - Sealed Class

  - Interfaces

Kotlin Classes – Normal Class

- As classes normais em Kotlin assim como em demais linguagens orientadas a objetos, permitem a definição de objetos com atributos (propriedades) e métodos (funções), possibilitando a reutilização e a organização do código

Kotlin Classes – Normal Class

- A declaração de classe consiste no nome da classe, no cabeçalho da classe (especificando seus parâmetros de tipo, o construtor primário e algumas outras coisas) e o corpo da classe entre chaves. Tanto o cabeçalho quanto o corpo são opcionais;
- Se a classe não tiver corpo, as chaves podem ser omitidas.

Dúvidas