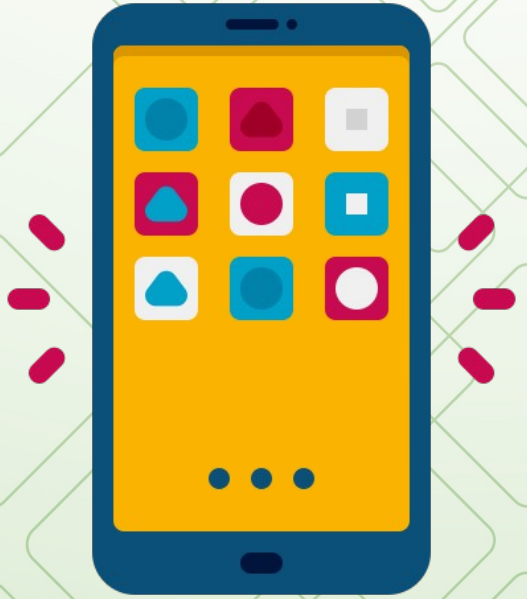
INSTITUTO FEDERAL
Santa Catarina

Programação para Dispositivos Móveis I



INSTITUTO FEDERAL
Santa Catarina

Linguagem de Programação

Kotlin

INSTITUTO FEDERAL
Santa Catarina

Tópicos

1) Introdução e sintaxe básica
2) Listas e Arrays
3) Condicionais e laços de repetição
4) Funções
5) Programação Orientada a Objetos

INSTITUTO FEDERAL
Santa Catarina

Introdução e sintaxe básica

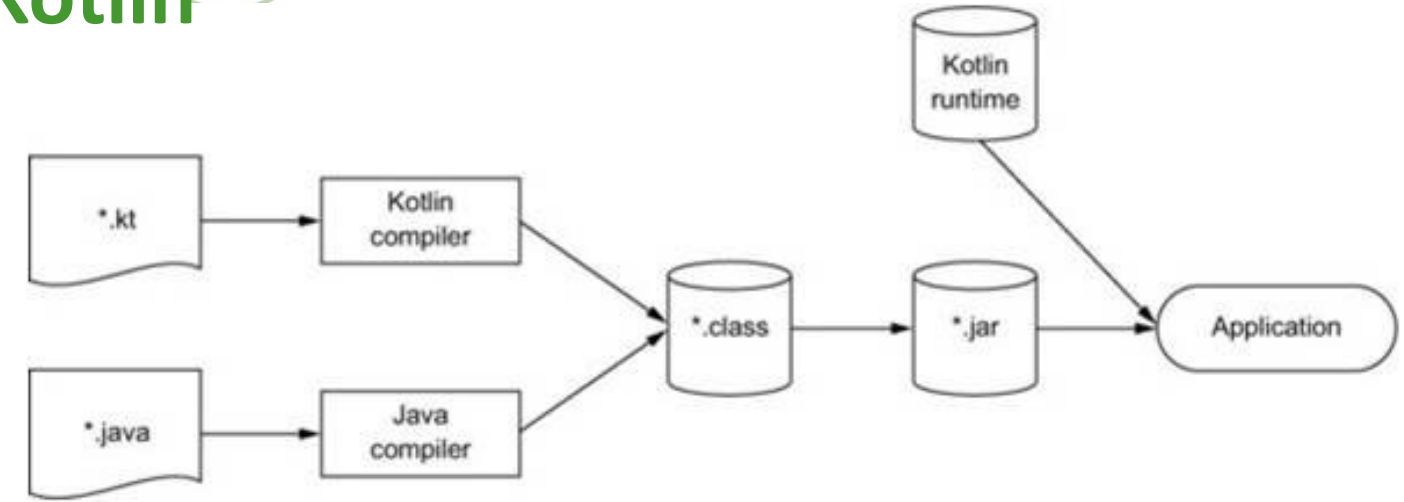
INSTITUTO FEDERAL
Santa Catarina

Kotlin

- Criada pela Jetbrains em 2011 como uma alternativa mais pragmática ao Java
- Kotlin é estaticamente tipada mas com suporte a inferência de tipos
- Suporte para programação orientada a objetos e programação funcional
- Gratuita e de código aberto

INSTITUTO FEDERAL
Santa Catarina

Kotlin



- Suporte completo para todos frameworks java, também rodando na JVM
- Possui recursos, como funções de extensão, expressões lambda e null safety
- Além de Android, permite programação lado servidor com recursos de geração de HTML
- play.kotl.in

INSTITUTO FEDERAL
Santa Catarina

Variáveis e constantes

- A definição dos tipos nas variáveis é opcional se for possível a inferência de tipos
- Os tipos primitivos funcionam também como classes
- **var** define uma variável
- **val** define uma constante
- Assim como os “if”, definição de variáveis é uma instrução e não uma expressão

- **Boolean**: representa valores lógicos, true ou false.
- **Byte**: representa valores inteiros de 8 bits.
- **Short**: representa valores inteiros de 16 bits.
- **Int**: representa valores inteiros de 32 bits.
- **Long**: representa valores inteiros de 64 bits.
- **Float**: representa valores de ponto flutuante de 32 bits.
- **Double**: representa valores de ponto flutuante de 64 bits.
- **Char**: representa valores de caracteres Unicode de 16 bits.

Variáveis e constantes

- “curso” é uma constante
- “nota” foi declarada e depois inicializada

```
fun main() {

    val curso = "CSTI"
    var ativa = false
    ativa = true
    var nota:Int
    nota = 10
    println(curso + " nota: " + nota)

}
// => CSTI nota: 10
```

Variáveis e constantes

- Existem constantes em tempo de execução (locais) e compilação (globais)
- As contantes globais precisam ser declaradas fora da função e ter a palavra reservada main.
- Sua verificação de valor é em tempo de compilação

```
const val nomeGlobal = "Antonio"

fun main() {

    val nomeLocal = "Miguel"
    println(nomeGlobal)
    println(nomeLocal)

}
// => Antonio Miguel
```

Variáveis e constantes

- Existem constantes em tempo de execução (locais) e compilação (globais)
- As contantes globais precisam ser declaradas fora da função e ter a palavra reservada main.
- Sua verificação de valor é em tempo de compilação

```
fun retornaNome()="Antonio"

const val nomeGlobal = "Miguel"

fun main() {

    val nomeLocal = retornaNome()
    println(nomeLocal)
    println(nomeGlobal)
}
// => Antonio Miguel
```

Variáveis e constantes

- Existem constantes em tempo de execução (locais) e compilação (globais)
- As contantes globais precisam ser declaradas fora da função e ter a palavra reservada main.
- Sua verificação de valor é em tempo de compilação

```
fun retornaNome()="Miguel"

const val nomeGlobal = retornaNome()

fun main() {

    val nomeLocal = "Antonio"
    println(nomeLocal)
    println(nomeGlobal)

}
// => erro
```

INSTITUTO FEDERAL
Santa Catarina

Template de Strings

Exemplo 1:

```
fun main() {

    val curso = "CSTI"
    var ativa = false
    ativa = true
    var nota:Int
    nota = 10
    println(curso + " nota: " + nota)
    // ou
    println("$curso nota: $nota")
    // utilize \$ para não imprimir como template

}
```

INSTITUTO FEDERAL
Santa Catarina

Template de Strings

Exemplo 2:

```
fun main() {

    val curso = "CSTI"
    var ativa = false
    ativa = true
    var nota:Int
    nota = 10

    println("$curso tem ${curso.length} letras")

}
// => CSTI tem 4 letras
```

Null Safety

- NullPointerException é uma exceção em tempo de execução em Java que ocorre quando uma variável é acessada mas não está apontando para nenhum objeto, se refere a nada ou null.
- Em Kotlin, a maioria dos tipos de dados não permitem valores nulos por padrão, o que significa que você não pode atribuir um valor nulo a uma variável ou passar um valor nulo como argumento para uma função, a menos que especifique explicitamente que o tipo permite nulos com “?”.

Exemplo 1:

```
fun main() {

    var nome: String = "Miguel"
    var sobrenome: String? = null
    println("$nome $sobrenome")

}
// => Miguel null
```

Null Safety

- Para evitar erro em tempo de execução, é possível fazer chamadas seguras utilizando “?”.
- O método “exibeNome()” causaria um erro caso fosse chamado em um objeto null.
- No caso ao lado, o método só será chamado se não for null.

Exemplo 2:

```
class Pessoa(nome:String, sobrenome:String){
    var nome = nome
    var sobrenome = sobrenome
    fun nomeCompleto():String{
        return "$nome $sobrenome"
    }
}

fun main() {
    val p = Pessoa("Miguel", "Zarth")
    //val p:Pessoa? = null
    print(p?.nomeCompleto() )
}
// => Miguel Zarth
```

Null Safety

- Para evitar error em tempo de execução, é possível fazer chamadas seguras utilizando “?”.
- O método “exibeNome()” causaria um erro caso fosse chamado em um objeto null.
- No caso ao lado, só será chamado se não for null.

Exemplo 2:

```
class Pessoa(nome:String, sobrenome:String){
    var nome = nome
    var sobrenome = sobrenome
    fun nomeCompleto():String{
        return "$nome $sobrenome"
    }
}

fun main() {
    //val p = Pessoa("Miguel", "Zarth")
    val p:Pessoa? = null
    print(p?.nomeCompleto() )
}
// => null
```

Null Safety

- Neste caso utilizamos o operador Elvis para um valor padrão.

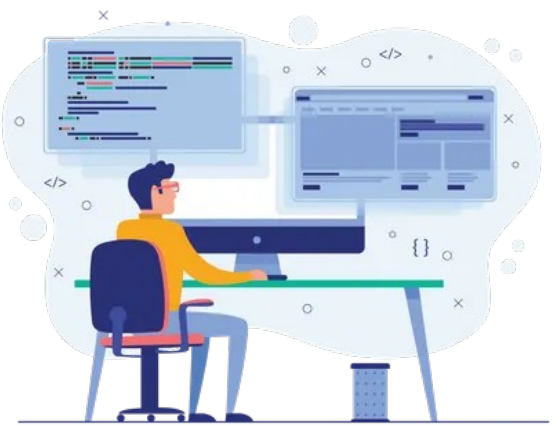
Exemplo 3:

```
class Pessoa(nome:String, sobrenome:String){
    var nome = nome
    var sobrenome = sobrenome
    fun nomeCompleto():String{
        return "$nome $sobrenome"
    }
}

fun main() {
    //val p = Pessoa("Miguel", "Zarth")
    val p:Pessoa? = null
    print(p?.nomeCompleto() ?: "anônimo")
}
// => anônimo
```

INSTITUTO FEDERAL
Santa Catarina

- Desenvolva um programa que solicite um dia e mês e informe se a data é válida.
- Ignore anos bissextos
- O mês deve ser lido com inteiros (1 a 12) mas a resposta deve ser por extenso. Exemplo:
  - “2 de janeiro é uma data válida”
- Utilizar *template strings*



INSTITUTO FEDERAL
Santa Catarina

Listas e Arrays

INSTITUTO FEDERAL
Santa Catarina

Lists

- **Listas** são coleções ordenadas de elementos
- Os elementos da lista podem ser acessados programaticamente através de seus índices
- Os elementos podem ocorrer mais de uma vez em uma lista
- Um exemplo de uma lista é uma frase: é um grupo de palavras, sua ordem é importante e elas podem repetir.

INSTITUTO FEDERAL
Santa Catarina

Immutable list utilizando listOf()

- Declare uma lista usando listOf() e imprima.

```
val instruments = listOf("trumpet", "piano", "violin")
println(instruments)
```

```
⇒ [trumpet, piano, violin]
```

INSTITUTO FEDERAL
Santa Catarina

Mutable list utilizando mutableListOf()

- As listas podem ser alteradas usando mutableListOf()

```
val myList = mutableListOf("trumpet", "piano", "violin")
 myList.remove("violin")
```

```
⇒ kotlin.Boolean = true
```

INSTITUTO FEDERAL
Santa Catarina

Arrays

- Arrays armazenam vários itens
- Os elementos de um Array podem ser acessados programaticamente por meio de seus índices
- Os elementos do Array são mutáveis
- O tamanho do Array é fixo

INSTITUTO FEDERAL
Santa Catarina

Arrays

- Uma Array de Strings pode ser criada usando arrayOf()
- Com um Array definida com **val**, você não pode alterar a qual variável se refere, mas ainda pode alterar o conteúdo do Array.

```
val pets = arrayOf("dog", "cat", "canary")
println(java.util.Arrays.toString(pets))
```

```
⇒ [dog, cat, canary]
```

INSTITUTO FEDERAL
Santa Catarina

Arrays

- Um Array pode ser de tipos diferentes ou mesmo tipo

```
val mix = arrayOf("hats", 2)

val numbers = intArrayOf(1, 2, 3)
```

INSTITUTO FEDERAL
Santa Catarina

Combinando Arrays

```
val numbers = intArrayOf(1,2,3)
val numbers2 = intArrayOf(4,5,6)
val combined = numbers2 + numbers
println(Arrays.toString(combined))
```

```
=> [4, 5, 6, 1, 2, 3]
```

INSTITUTO FEDERAL
Santa Catarina

Condicionais e laços de repetição

INSTITUTO FEDERAL
Santa Catarina

Condicionais

Exemplo 1:

```
val numberOfCups = 30
val numberOfPlates = 50

if (numberOfCups > numberOfPlates) {
    println("Too many cups!")
} else {
    println("Not enough cups!")
}
```

=> Not enough cups!!

INSTITUTO FEDERAL
Santa Catarina

Condicionais

Exemplo 2:

```
val guests = 30
if (guests == 0) {
    println("No guests")
} else if (guests < 20) {
    println("Small group of people")
} else {
    println("Large group of people!")
}
```

```
=> Large group of people!
```

INSTITUTO FEDERAL
Santa Catarina

Condicionais

Exemplo 3:

```
val numberOfStudents = 50
if (numberOfStudents in 1..100) {
    println(numberOfStudents)
}
```

```
=> 50
```

INSTITUTO FEDERAL
Santa Catarina

Condicionais

Exemplo 4:

```
val results = 59

when (results) {
    0  -> println("No results")
    1  -> println("Only one")
    in 2..39 -> println("Got results!")
    else -> println("That's a lot of results!")
}
```

=> That's a lot of results!

INSTITUTO FEDERAL
Santa Catarina

For Loops

```
val pets = arrayOf("dog", "cat", "canary")
for (element in pets) {
    print(element + " ")
}
```

```
=> dog cat canary
```

INSTITUTO FEDERAL
Santa Catarina

For Loops

```
for ((index, element) in pets.withIndex()) {
    println("Item at $index is $element\n")
}
```

```
=> Item at 0 is dog
   Item at 1 is cat
   Item at 2 is canary
```

INSTITUTO FEDERAL
Santa Catarina

For Loops

```
for (i in 1..5) print(i)

for (i in 5 downTo 1) print(i)

for (i in 3..6 step 2) print(i)

for (i in 'd'..'g') print (i)
```

```
=> 12345
=> 54321
=> 35
=> defg
```

INSTITUTO FEDERAL
Santa Catarina

while loops

```
var bicycles = 0
while (bicycles < 50) {
    bicycles++
}
println("$bicycles bicycles in the bicycle rack\n")

do {
    bicycles--
} while (bicycles > 50)
println("$bicycles bicycles in the bicycle rack\n")
```

```
⇒ 50 bicycles in the bicycle rack
⇒ 49 bicycles in the bicycle rack
```

INSTITUTO FEDERAL
Santa Catarina

repeat loops

```
repeat(2) {
    print("Hello!")
}
```

⇒Hello!Hello!

INSTITUTO FEDERAL
Santa Catarina

Funções

INSTITUTO FEDERAL
Santa Catarina

Sobre as funções

- Um bloco de código que executa uma tarefa específica
- Divide um programa grande em pedaços modulares menores
- Declarado usando a palavra-chave **fun**
- Pode usar argumentos com valores nomeados ou padrão

```
fun printHello() {
    println("Hello World")
}
```

INSTITUTO FEDERAL
Santa Catarina

Sobre as funções

- Se uma função não retornar nenhum valor útil, seu tipo de retorno será Unit.
- Sua declaração é opcional

```
fun printHello(name: String?): Unit {
    println("Hi there!")
}
// ou
fun printHello(name: String?) {
    println("Hi there!")
}
```

INSTITUTO FEDERAL
Santa Catarina

Parâmetros default

```
fun drive(speed: String = "fast") {
    println("driving $speed")
}
drive() // ⇒ driving fast
drive("slow")  // ⇒ driving slow
drive(speed = "turtle-like") // ⇒ driving turtle-like
```

INSTITUTO FEDERAL
Santa Catarina

Parâmetros requeridos

```
fun tempToday(day: String, temp: Int) {
    println("Today is $day and it's $temp degrees.")
}
```

INSTITUTO FEDERAL
Santa Catarina

Parâmetros requeridos

```
fun tempToday(day: String, temp: Int) {
    println("Today is $day and it's $temp degrees.")
}

fun reformat(str: String,
  divideByCamelHumps: Boolean,
             wordSeparator: Char,
             normalizeCase: Boolean = true){
...
}
```

não

Single-expression functions

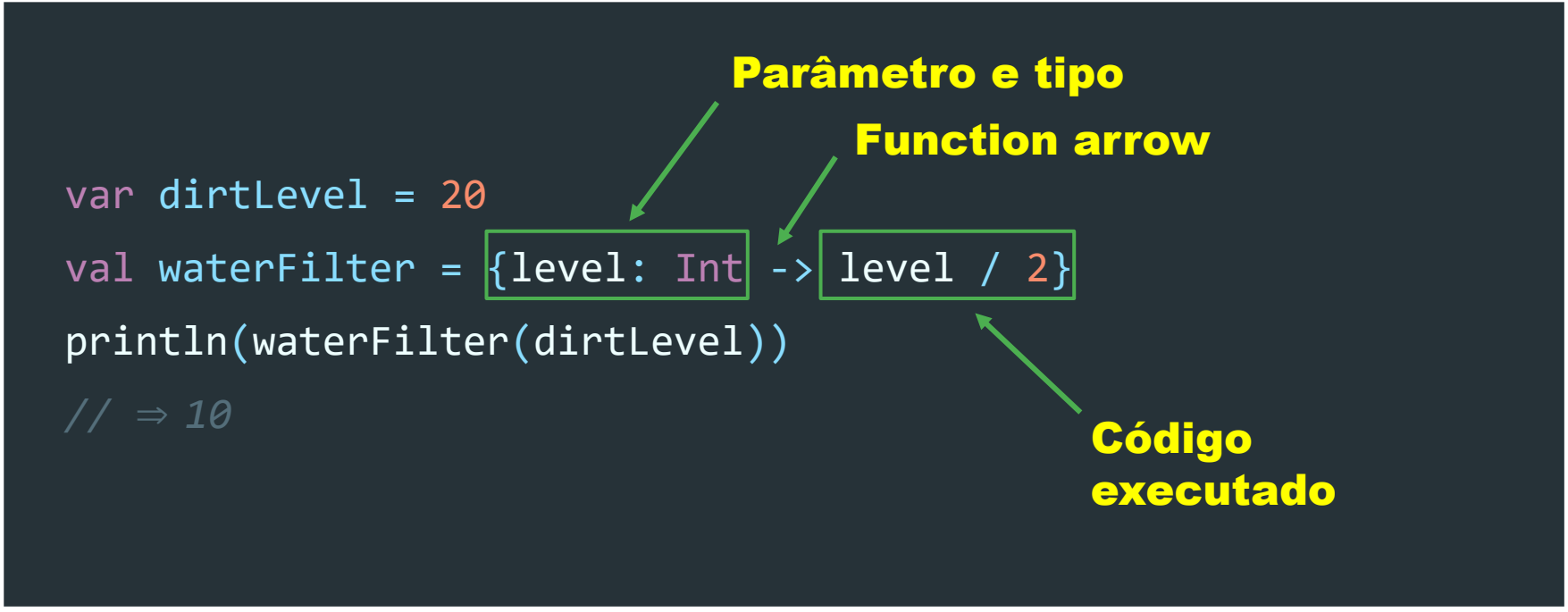
```
fun double(x: Int): Int {
    x * 2
}

fun double(x: Int):Int = x * 2
```

Versão compacta

Funções lambda

Um lambda é uma expressão que cria uma função que não tem nome.



INSTITUTO FEDERAL
Santa Catarina

Higher-order functions

```
fun encodeMsg(msg: String, encode: (String) -> String): String {
    return encode(msg)
}

val enc1: (String) -> String = { input -> input.toUpperCase() }

println(encodeMsg("abc", enc1))
```

INSTITUTO FEDERAL
Santa Catarina

Higher-order functions

```
fun encodeMsg(msg: String, encode: (String) -> String): String {
    return encode(msg)
}

fun enc2(input:String): String = input.reversed()
// passar como parâmetro uma função com nome em vez de uma lambda:
encodeMessage("abc", ::enc2)
```

- Faça 4 funções nomeadas relativas as operações básicas: somar, subtrair, dividir e multiplicar.
- Faça uma função lambda de potênciação
- Faça uma função “calcular” que recebe 2 parâmetros double e uma função específica (somar, dividir, subtrair, multiplicar, potência)
- Teste calcular() executando as 5 operações