Aula 2

Programação IV

Profª Ana Paula Costacurta

1

Conversa Inicial

2

Tópicos

- Introdução ao Kotlin
- Sintaxe básica e declaração de variáveis
- Funções
- Classe e orientação a objeto
- Kotlin avançado

3

Introdução ao Kotlin

4

Kotlin

- Criada pela JetBrains → Android Studio
- Compilada para Executar na JVM
  - Pode usar classes escritas em java
- Escrever código com menos linhas
  - Bibliotecas e plugins → simplificar o código

5

Java vs Kotlin

- Exemplo tratar evento clique no Botão:
  - Java:

```
View btAgendar = findViewById(R.id.btAgendar);
btAgendar.setOnClickListener(new View.OnClickListener() {
      @Override public void onClick(View v) {
            agendar();
      } }
);
```

  - Kotlin:

```
findViewById<Button>(R.id.btAgendar).setOnClickListener(
   agendar()
)
```

6

Kotlin Playground

- Ferramenta on-line
- Com acessado pelo navegador:
  - Link: <https://play.kotlinlang.org/>
- Criado para se acostumar com a codificação sem a necessidade de instalar SW
- Já abre com a função main() com código para apresentar na tela "Hello, world!!!"
- Clica em "Run" para executar o programa

7

Sintaxe básica e declaração de variáveis

8

String

- Declaração de variável:
  - var «nome»:String = "«Texto»"
  - var «nome» = "«Texto»" → atribuição de valor
- Declaração de constante:
  - val «nome» = "«Texto»"
- Referência ou expressões:
  - ${«nome»}
  - ${«objeto».«propriedade»}

9

Lista

- Declaração:

var nomes:ArrayList<String> = ArrayList<String>()

var nomes = ArrayList<String>()

- Adicionar item:

nomes.add("nome")

- Criando com inclusão:

val nomes:List<String> = listOf("nome1", "nome2", "nome3")

10

Conversões de tipo

- Smart Cast → is
  - «variavel» is String
  - verifica e converte automaticamente
- Cast → as
  - «variavel» as String
  - exceção do tipo ClassCastException
- Cast seguro → as?
  - «variavel» as String
  - Se não for string retorna null

11

Operadores

- Evitar o uso do if/else:
  - Ternário:
    - Java → (a == 5) ? "sim" : "não"
    - Kotlin → (a == 5) "sim" else "não"
  - Elvis
    - Se titulo não for null → retorna ele
    - caso contrário → retorna "Olá!"
    - título?: "Olá!"

```
if ($a == 5)
   echo 'sim';
else
   echo 'não';
```

12

Funções

13

Declaração

```
fun test(param1: Tipo, param2: Tipo, ...): TipoRetorno {
      // Código
}
```

- **TipoRetorno Unit → sem retorno (opcional)**
- **Tipoparametro? → Aceita null**
- **Função em uma linha → substitui "{}" por "="**

```
fun imprimir(s: String) {
      println(s)
}
```

------→

```
fun imprimir(s: String) = println(s)
```

14

Default arguments

- **Evitar criar vários métodos com a mesma assinatura**
- **Parâmetros com valores padrões → utilizado caso não seja informado**

```
fun nomefuncao(parm1: Tipo, param2:Tipo = "Texto"): Tipo Retorno {
       return numero
}
```

15

Named arguments

- **Passagem de Parâmetros fora de ordem;**
- **Passagem de parâmetros por nome;**
- **Sintaxe:**

```
fun nomefuncao(parm1: Tipo, param2:Tipo? = "Texto", param2: Tipo, param3: tipo): Tipo Retorno {
       return numero
```

- **Chamada:**

```
println(nomefuncao(parm2="texto")
```

16

varargs

- **1 ou + parâmetros separados por ","**
- **Sintaxe:**

```
fun nomefuncao(vararg args: String): List<String> {
```

- **Chamada:**

```
var nome = nomefuncação("parm1","parm2","parm3")
```

17

Tipos Genéricos <T>

- **Cria uma função que pode ser de qualquer tipo**
- **Algoritmo genérico para qualquer tipo**
- **Sintaxe:**

```
fun <T> toList(vararg args: T): List<T> {
   val list = ArrayList<T>()
```

- **Chamada:**

```
val strings = toList<String>("Banana","Laranja","Maçã")
```

18

Classes e orientação a objeto

19

Conceitos

- Classes → modelo de objetos (dados e comportamentos)
- Interfaces → métodos declarados (o que fazer)
- Métodos → comportamentos
- Atributos → dados
- Construtor → criação do objeto
- Herança → classes com coisas em comum
- Encapsulamento → esconder, usando get e set

20

Classe e construtor

- Classe → nome, cabeçalho (parâmetros de tipo, construtor padrão e em algumas outras coisas):

```
class Pessoa { /*corpo*/ }
class Pessoa
```

- Construtor primário→ na mesma linha

```
class Person constructor(firstName: String) { /*...*/ }
class Person (firstName: String) { /*...*/ }
```

- Instância → sem palavra "new":

```
Pessoa ()    ou    Pessoa("João")
```

21

Herança

- Utilizar ":nome da classe-mãe"
- Precisa estar como "open" → padrão final

```
open class Automovel(nome:String, ano:Int) {
        val nome: String
        val ano: Int
         /* Corpo do classe*/
}


class Carro(nome:String, ano:Int): Automovel(nome,ano) {
         /* Corpo do classe*/
}
```

22

Classe de Dados (Data Classes)

- Data Classes → : manter dados
  - implementam as seguintes funções automaticamente:
    - equals() e hashcode()
    - toString()
    - copy()

```
data class Carro(val nome:String)
```

23

Método estático e Singletons

- Objeto companion:
  - Método estático → declaração dentro do objeto companion
  - Elimina a necessidade: criar instância, salvar o estado do objeto
- Singletons:
  - uma classe tenha apenas uma instância (objeto) em memória
  - Trocar class → object

24

Listas – listOf e mutableListOf

- Duas maneiras de criar listas:
  - Arraylist:

```
val list = ArrayList<String>()
```

  - MutableListOf:

```
val carros = mutableListOf<Carro>()
```

- Com mutableListOf:
  - Implementação abstraída
  - ListOf → imutável
  - É a recomendada

25

Classes Enum

- Contêm constantes predefinidas, exemplo Estados, direções etc.
- Construtor:

```
enum class Color(val rgb: Int) {
    RED(0xFF0000),
    GREEN(0x00FF00),
    BLUE(0x0000FF)
}
```

- Acesso da cor:

```
val azul = Color.BLUE.rgb
```

26

Kotlin avançado

27

Higher-Order Functions e Lambdas

- Funções recebam outras funções no parâmetro
- Função retorna outra função
- A sintaxe → "::funcao" `val pares = filtrar(ints, ::numerosPares)`
- Lambdas:
  - Forma simplificada de passar como parâmetro
  - Redução da quantidade de código escrito
  - Função recebe apenas um parâmetro → it

```
val pares = filtrar(ints, {numerosPares(it)})
```

28

Lambdas → funções anônimas

- Podemos passar funções como parâmetros sem criar essas funções

```
val pares = filtrar(ints, {numerosPares(it)})
```

↓

```
val pares = filtrar(ints) { it % 2 == 0 }
```

29

Extensões

- Criar um método em uma classe final

```
fun classe.novo_metodo(): <tipo> { . . . }
```

30

Coleções e lambdas: map e filter

- Funções nativas -> processar e filtrar listas, sets, mapas, dentre outras coisas
- Vamos ver:
  - map
  - filter

31

Objetos nulos (Null Safety)

- Kotlin não permite que variáveis e objetos tenham valores nulos
- O código não compila, pois a variável pode ser nula
- Variável pode ter um valor nulo:

```
var nome:String? = "Steve"
```

- safe call (chamada segura)
  - vai ignorar a chamada se o objeto for nulo
  - Sintaxe: ${variavel?.metodo}

32

NullPointerException

- O recomendado é sempre validar as variáveis se não são nulas ou utilizar o operador ?
- Podemos forçar que a referência de um objeto seja acessada com operador !!
- Evitar ao máximo a utilização:
  - Sintaxe: {variavel!!.metodo}

33

34