

Desenvolvimento para Dispositivos Móveis

Programação Kotlin II

Prof. Chauã Queirolo

Sumário

- Estruturas de decisão
- Estruturas de repetição
- Definição de funções
- Programação orientada a objetos


Estruturas de decisão

Estruturas de decisão

- As estruturas de decisão são:
 - `if .. else`
 - `if .. else if`
 - `when`

Estruturas de decisão


if .. else



```
if (numero % 2 == 0) {  
    println("$numero é par")  
} else {  
    println("$numero é ímpar")  
}
```

Estruturas de decisão


if .. else if



```
if (numero > 0) {  
    println("$numero é positivo")  
} else if (numero < 0) {  
    println("$numero é negativo")  
} else {  
    println("$numero é neutro")  
}
```

Estruturas de decisão

when



```
when (operador) {  
    "+" -> println("$a + $b = ${a + b}")  
    "-" -> println("$a - $b = ${a - b}")  
    "*" -> println("$a * $b = ${a * b}")  
    "/" -> println("$a / $b = ${a / b}")  
    else -> println("Operador inválido")  
}
```

Estruturas de decisão

inline



```
val tipo = if (numero % 2 == 0) "par" else "ímpar"
```



Estruturas de repetição

Estruturas de repetição

- As estruturas de repetição são:
 - `while`
 - `do .. while`
 - `for`

Estruturas de repetição

while




```
var numero = 0

while (numero < 10) {
    print("$numero, ")
    numero += 1
}
```

Estruturas de repetição

do .. while




```
var valor: String

do {
    println("Digite 'sair' para sair: ")
    valor = readLine()!!
} while (valor != "sair")
```

Estruturas de repetição

for



```
for (i in 1..10) {  
    print("$i, ")  
}
```

Estruturas de repetição


Controle de fluxo

- Para alterar o fluxo de execução de um laço:
 - `break`
 - `continue`

Definição de funções

Definição de funções

Declaração




```
fun imprime1() {  
    println("1")  
}
```

```
fun imprime5() = println("5")
```

```
fun get7() = 7
```


Definição de funções


Declaração



```
fun soma(a: Int, b: Int): Double {  
    val soma = a + b  
    return soma.toDouble()  
}
```

Definição de funções

Parâmetros padrão



```
fun imprimeIntervalo(min: Int = 0, max: Int = 10) {  
    for (i in min..max) {  
        print("$i, ")  
    }  
    println()  
}
```

Definição de funções

Parâmetros nomeados



```
imprimeIntervalo(7, 15)
```

```
imprimeIntervalo(7)
```

```
imprimeIntervalo()
```

```
imprimeIntervalo(min = 10, max = 20)
```

Paradigmas de programação

Paradigmas de programação

Introdução

- **Visão** do **programador** em relação aos **programas**
 - Estrutura
 - Execução
- Principais **paradigmas**:
 - **Imperativo**: estruturado, procedural, orientado a objetos
 - **Declarativo**: funcionalista, lógico

Paradigmas de programação

Paradigma orientado a objetos

- **Descreve** o sistema com elementos do **mundo real**
- **Considera** que todas as **componentes** são **objetos**
- **Objetos** possuem sua **estrutura** e desempenham **ações**
- **Classificados** de acordo com suas **características**
- **Exemplo:** Java, C++, C#, Python

Paradigmas de programação

Paradigma orientado a objetos

- **Vantagens**

- Abstração
- Modularização
- Extensibilidade
- Reaproveitamento de código

Orientação a objetos

Objetos

Conceitos

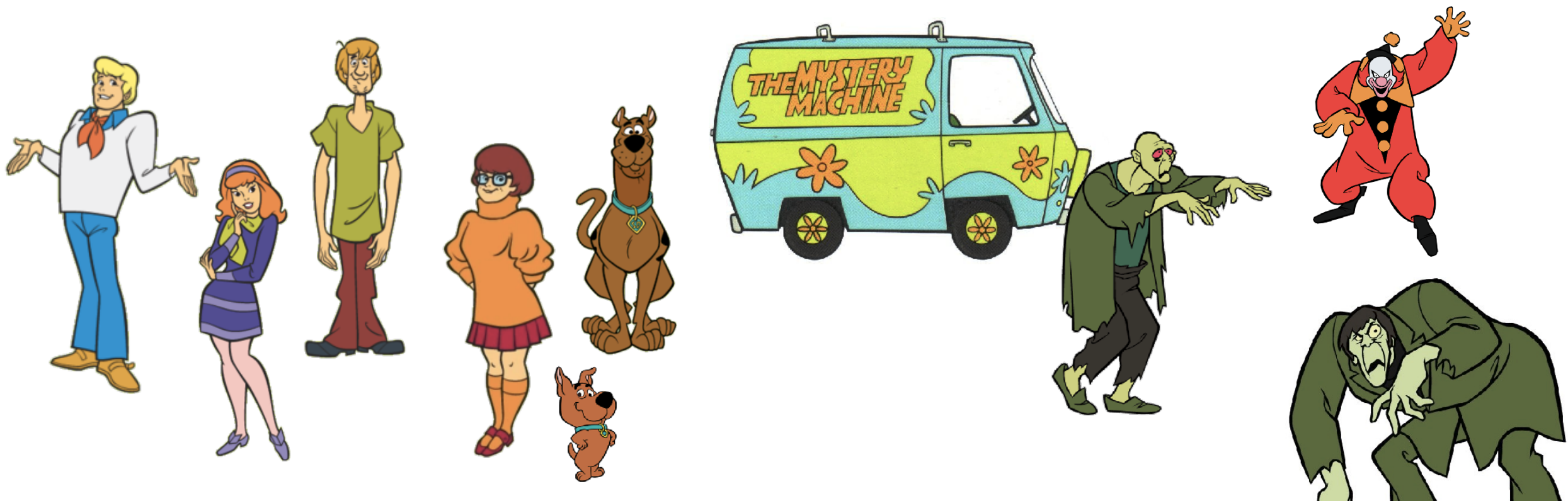
- O **universo** é formado por **objetos**
- Cada **objeto** possui:
 - **Características**
 - **Funções**



Classes

Conceitos

- Os **objetos** são **classificados** de acordo com
 - **Características** semelhantes
 - **Funcionalidades** semelhantes
- **Classes** são **agrupamentos** objetos **semelhantes** entre si



Classes

Conceitos

- **Atributos:** características, propriedades
- **Métodos:** funcionalidades, ações, procedimentos

Pessoa
<ul style="list-style-type: none">- nome- idade- peso- altura
<ul style="list-style-type: none">+ andar()+ conversar()+ dormir()+ dirigirCarro()



Classes

Relacionamentos

- Associação
- Agregação
- Composição
- Herança

Pilares da Orientação a Objetos

- Encapsulamento
- Herança
- Polimorfismo

Encapsulamento

- Todo objeto é responsável pelos seus atributos
- Esconder do mundo externo:
 - Estrutura interna dos objetos
 - Detalhes de implementação
- Interação através de uma interface pública

Herança

- Estabelece a relação **é um** entre duas classes
- Permite
 - Abstração
 - Reaproveitamento de código

Polimorfismo

- Um objeto pode assumir **diferentes formas**
- **Tipos** de polimorfismo:
 - **por inclusão** - via herança
 - **paramétrico** - tipos genéricos
 - **sobrescrita** - redefinição de métodos
 - **sobrecarga** - métodos com mesmo nome, parâmetros diferentes

Programação orientada a objetos

Referências bibliográficas



Referência Bibliográficas

- **Linguagem Kotlin:** <https://kotlinlang.org/>
- **Tutorial Kotlin:** <https://www.programiz.com/kotlin-programming>