Desenvolvimento para Dispositivos Móveis

Variáveis, Operadores e Estruturas Condicionais

Prof. Bruno Azevedo

UNIP – Universidade Paulista

Informação

- Matéria-prima processada pelos computadores.
- A informação pode ser representada por diferentes tipos primitivos.
- Por exemplo, a idade de uma pessoa pode ser representada por um número inteiro (em anos).
- A altura de uma pessoa pode ser representada por um número real (em metros).

Tipos Primitivos

- Inteiro: informação numérica que pertença ao conjunto dos números inteiros.
  - Ele tem 15 irmãos.
  - A temperatura desta noite será de:2 graus celsius.
- Real: informação numérica que pertença ao conjunto dos números reais.
  - Ele tem 1,73 metros de altura.
  - Meu saldo bancário é de R$92,17.
- String: informação composta por um ou mais caracteres alfanuméricos. Uma `sequência de caracteres`.
  - Meu nome começa com a letra “A”.
  - A placa dizia: “Não pise na grama”.
- Booleana: informação que pode assumir apenas dois estados lógicos estáveis e opostos; geralmente verdadeiro ou falso.
  - A informação de que eu tenho um cachorro é falsa.

Tipos Primitivos

Qual o tipo primitivo da informação destacada nas afirmações abaixo?

- Ângelo ficou devendo `R$53,12` reais ao vendedor.
- José subiu `cinco` degraus, desistiu, e desceu da escada.
- Marcos venceu a corrida com o tempo de `57,3` segundos.
- O outdoor dizia `"Estude na Universidade Paulista!"`.

Variáveis e Constantes

- As informações podem ser constantes ou variáveis.
- Existem informações que nunca mudam, ou que não queremos que mudem.
- Existem informações que são mutáveis, ou que podemos querer mudar.

Constantes

- Uma informação é constante quando não sofre variação com o decorrer do tempo; não pode ser alterada; valor fixo, sempre possui o mesmo valor.
  - O valor de π é uma constante de 3.14159...
  - A velocidade da luz é uma constante de 299.792.458 m/s.
- Se um valor não muda, ou não desejamos que ele mude (em programação), ele é constante.

Variáveis

- Uma informação é variável quando pode mudar, em algum instante, com o decorrer do tempo.
  - O seu peso pode variar durante a sua vida
  - A sua idade varia a cada ano.
- Se um valor pode mudar com o tempo, ele é variável.

Introdução ao Desenvolvimento em Kotlin

- Kotlin é uma linguagem de programação moderna, adotada como a linguagem oficial para o desenvolvimento de aplicativos Android pelo Google.
  - Kotlin é 100% compatível com Java, ou seja, é possível usar bibliotecas Java diretamente no Kotlin.
  - Combina aspectos da programação funcional e orientada a objetos.
- Kotlin foi criado pela JetBrains e se tornou uma linguagem oficial para desenvolvimento Android em 2017.
- ▷ Exemplo simples em Kotlin:

```
fun main() {
    val mensagem = "Olá, Kotlin!"
    println(mensagem)
}
```

Tipos de Variáveis em Kotlin

- Em Kotlin, existem diferentes tipos de dados que podem ser atribuídos às variáveis.
- Alguns dos tipos comuns são:
  - `Int`: Para números inteiros (exemplo: 1, 2, 100).
  - `Double`: Para números reais (exemplo: 1.5, 3.14, 0.99).
  - `String`: Para texto ou sequências de caracteres (exemplo: “Olá”, “Kotlin”).
  - `Boolean`: Para valores lógicos, como `true` ou `false`.
  - `Char`: Para caracteres únicos (exemplo: ’A’, ’b’).
- Exemplo de variáveis em Kotlin:

```
val idade: Int = 25
val altura: Double = 1.65
val nome: String = "Ana"
val ativo: Boolean = true
val letra: Char = 'K'
```

- Atributos e valores podem ser diferentes, conforme o tipo. Cada tipo tem seu próprio comportamento e uso.

Mutabilidade: `val` vs `var`

- Em Kotlin, as variáveis podem ser classificadas como imutáveis ou mutáveis.
  - `val`: A variável é imutável, ou seja, o valor não pode ser alterado após a atribuição inicial.
  - `var`: A variável é mutável, permitindo que o valor seja alterado durante a execução do programa.
  - ▷ Exemplo:

```
val nomeImutavel: String = "Alice"    // Não pode ser alterado
var idadeMutavel: Int = 25            // Pode ser alterado
```

- A recomendação é usar `val` sempre que possível, já que isso oferece mais segurança e previsibilidade no código.

Declaração Explícita de Tipos

- Em Kotlin, podemos declarar explicitamente o tipo de uma variável.
- Isso é útil quando queremos garantir que a variável tenha um tipo específico, especialmente quando o tipo não é óbvio.
- ▷ Exemplo de declaração explícita:

```
val idade: Int = 30          // Tipo explícito: Int
val preco: Double = 100.50   // Tipo explícito: Double
val nome: String = "Carlos" // Tipo explícito: String
val ativo: Boolean = true    // Tipo explícito: Boolean
```

- Ao usar a declaração explícita de tipos, você deixa claro qual é o tipo da variável e o compilador pode verificar se o tipo é compatível com o valor atribuído.

Declaração Implícita de Tipos

- Quando você omite o tipo de uma variável, o Kotlin pode inferir automaticamente o tipo com base no valor atribuído.
- ▷ Exemplo de declaração implícita:

```
val idade = 30       // Kotlin infere que é um Int
val preco = 100.50   // Kotlin infere que é um Double
val nome = "Carlos" // Kotlin infere que é uma String
val ativo = true     // Kotlin infere que é um Boolean
```

- A declaração implícita torna o código mais conciso, mas, ao mesmo tempo, o compilador garante que o tipo seja corretamente inferido.

Atribuição de Valores a Variáveis

- A atribuição de valores em Kotlin é feita utilizando o operador de atribuição =.
  - Para `val`, a variável recebe o valor uma única vez, na declaração, sendo imutável posteriormente.
  - Para `var`, a variável pode ser reatribuída com novos valores durante a execução do código.

▷ Exemplo de atribuição:

```
val nome = "João" // Atribuindo valor à variável imutável nome
var idade = 30    // Atribuindo valor à variável mutável idade
idade = 31        // Reatribuindo valor à variável idade
```

- A variável `nome` não pode ser alterada porque foi declarada com `val`.
- A variável `idade` pode ser alterada pois foi declarada com `var`.

Exemplo Completo

- Vamos ver um exemplo de código Kotlin que faz uso de variáveis mutáveis e imutáveis.

```
fun main() {
    val nome = "Maria" // nome é imutável
    var idade = 30     // idade é mutável
    println("Nome: $nome")
    println("Idade: $idade")
    idade = 31  // Modificando a idade
    println("Idade atualizada: $idade")
}
```

- Saída:

```
Nome: Maria
Idade: 30
Idade atualizada: 31
```

Função `println` em Kotlin

- A função `println` em Kotlin é utilizado para imprimir informações no console.
- Exibe a mensagem seguida por uma quebra de linha.
- ▷ Exemplo de uso simples:

```
fun main() {
    println("Olá, Kotlin!")  // Imprime o texto na tela
}
```

- Saída:

```
Olá, Kotlin!
```

Função `readLine()` em Kotlin

- A função `readLine()` é usada para ler dados do teclado.
- Ela retorna uma `String`, por isso muitas vezes é necessário tratar ou converter o valor lido.
- Para ler números, usamos funções como `toInt()`, `toDouble()`, etc., após o uso de `readLine()`.

```
print("Digite seu nome: ")
val nome = readLine()
print("Digite sua idade: ")
val idade = readLine()?.toInt()
println("Olá, $nome! Você tem $idade anos.")
```

- A interrogação (?) após o `readLine()` é o operador de chamada segura.
- Resumindo, estamos dizendo "só chama o toInt() se o valor não for nulo".
- Se o usuário digitar 25, Ok. Mas se der Null por algum motivo, não teremos uma exceção NullPointerException.

Criação de um `Array`

- Em Kotlin, um `Array` é uma estrutura de dados que pode armazenar elementos de um mesmo tipo.
- A criação de um `Array` é feita utilizando a função `arrayOf()` ou a função `Array()` para arrays de tamanho fixo.
- O tipo dos elementos é inferido automaticamente ou pode ser especificado explicitamente.
  - Exemplo com definição implícita de tipo:

```
val arrayExemplo = arrayOf(1, 2, 3)
```

  - Exemplo com definição explícita de tipo:

```
val arrayExemplo: Array<Int> = arrayOf(1, 2, 3)
```

Alterando Elementos em um `Array` em Kotlin

- Em Kotlin, os elementos de um `Array` podem ser alterados diretamente, pois o conteúdo de um `Array` é mutável, mas o tamanho não pode ser alterado após a criação.
- Para alterar um elemento, basta acessar a posição desejada e atribuir um novo valor.
- ▷ Exemplo:

```
val arrayExemplo = arrayOf(1, 2, 3)
arrayExemplo[0] = 10  // Alterando o primeiro elemento
```

- Atenção: O `Array` é indexado em zero.

Criação de uma `MutableList`

- Um `List` em Kotlin é uma coleção que armazena múltiplos elementos.
- Existem dois tipos principais de listas: mutáveis (`MutableList`) e imutáveis (`List`).
- Vamos demonstrar a criação de uma lista mutável utilizando a função `mutableListOf()`.
- ▷ Exemplo de criação de um `MutableList`:

```
val listaMutavel = mutableListOf("Maria", "João")
```

- A `MutableList` permite que você adicione, remova e altere os elementos da lista.
- Para listas imutáveis, você usaria `listOf()` em vez de `mutableListOf()`.

Adicionando Elementos em uma MutableList

- O MutableList permite que você altere dinamicamente seu conteúdo.
- Para adicionar um elemento, podemos usar a função add().
- ▷ Exemplo de adição de elementos:

```
val mutableList = mutableListOf(1, 2, 3)
mutableList.add(4)  // Adicionando um novo elemento ao final
println(mutableList)  // Exibe: [1, 2, 3, 4]
mutableList.add(1, 5)  // Adicionando 5 na posição 1
println(mutableList)  // Exibe: [1, 5, 2, 3, 4]
```

Modificando Elementos em uma MutableList

- Para modificar um elemento, basta acessar o índice desejado e atribuir um novo valor.
- ▷ Exemplo de modificação de um elemento:

```
val mutableList = mutableListOf(1, 2, 3)
mutableList[0] = 10  // Alterando o primeiro elemento
println(mutableList)  // Exibe: [10, 2, 3]
```

Removendo Elementos de uma MutableList

- O MutableList também permite remover elementos, tanto pelo valor quanto pela posição.
- ▷ Exemplo de remoção pelo valor:

```
val mutableList = mutableListOf(1, 2, 3, 4)
mutableList.remove(3)  // Removendo o valor 3
println(mutableList)  // Exibe: [1, 2, 4]
```

remove() remove o primeiro elemento encontrado com o valor especificado.

- ▷ Exemplo de remoção pela posição:

```
val mutableList = mutableListOf(1, 2, 3, 4)
mutableList.removeAt(1)  // Removendo o elemento na posição 1 (valor 2)
println(mutableList)  // Exibe: [1, 3, 4]
```

Criação de um Pair

- O Pair em Kotlin é uma estrutura de dados que armazena dois valores, que podem ser de tipos diferentes.
- Útil quando precisamos retornar ou armazenar dois valores juntos.
- ▷ Exemplo de criação de um Pair:

```
val pessoa = Pair("Joana", 22)  // Pair de String e Int
println(pessoa.first)  // Acessando o primeiro valor
println(pessoa.second) // Acessando o segundo valor
println("Nome: ${pessoa.first}, Idade: ${pessoa.second}")
```

- O Pair em Kotlin é imutável. Ou seja, uma vez criado, você não pode alterar diretamente os valores armazenados.

Criação de uma `MutableList` Inicial Vazia

- Podemos escolher criar uma lista vazia e adicionar elementos posteriormente:
  - ▷ Exemplo de criação de um `MutableList` onde cada elemento é um `Pair` de `String` e `Int`:

```
val pessoas = mutableListOf<Pair<String, Int>>()
```

- Após a criação da lista, podemos adicionar elementos do tipo `Pair<String, Int>` à lista.

```
pessoas.add(Pair("Maria", 30))
pessoas.add(Pair("João", 25))
pessoas.add(Pair("Ana", 28))
```

Operadores Aritméticos

- Os operadores aritméticos são usados para realizar cálculos matemáticos básicos.
- Os principais operadores aritméticos em Kotlin são:
  - +: Adição
  - -: Subtração
  - *: Multiplicação
  - /: Divisão
  - %: Resto da divisão inteira (Módulo)
- Exemplo:

```
val a = 9
val b = 3
println(a + b)  // Soma
println(a - b)  // Subtração
println(a * b)  // Multiplicação
println(a / b)  // Divisão
println(a % b)  // Módulo
```

- Saída:

```
12
6
27
3
```

Operadores Relacionais

- Os operadores relacionais são usados para comparar valores e retornam um valor booleano (`true` ou `false`).
- Os principais operadores relacionais em Kotlin são:
  - `>`: Maior que
  - `<`: Menor que
  - `==`: Igual a
  - `!=`: Diferente de
- Exemplo:

```
val x = 5
val y = 10
println(x > y)
println(x < y)
println(x == y)
println(x != y)
```

- Saída:

```
false
true
false
true
```

Operadores Lógicos

- Os operadores lógicos são usados para combinar expressões booleanas.
- Os principais operadores lógicos em Kotlin são:
  - `&&`: E lógico (and)
  - `||`: Ou lógico (or)
  - `!`: Negação (not)
- Exemplo:

```
val x = true
val y = false
println(x && y)
println(x || y)
println(!x)
```

- Saída:

```
false
true
false
```

Combinando Condições

- Os operadores lógicos `&&` e `||` permitem combinar múltiplas condições em uma expressão.
- `&&`: Retorna `true` se ambas as condições forem verdadeiras.
- `||`: Retorna `true` se pelo menos uma das condições for verdadeira.
- ▷ Exemplo:

```
val a = 5
val b = 10
val c = 20
println(a < b && b < c)
println(a > b || b < c)
```

- Saída:

```
true
true
```

Praticando o Desenvolvimento em Kotlin

- Inicialmente, utilizaremos o One Compiler `https://onecompiler.com/kotlin/` para praticar o desenvolvimento em Kotlin.

Atividade Prática 2

1. Criar uma lista com nomes de pessoas:
   - Inclusão de nomes na lista.
   - Remoção de nomes da lista.
2. Ampliação: Alterar as informações para conter um conjunto nome, idade.
3. Cada aluno deve produzir um relatório de 1 a 3 páginas contendo:
   - Resumo teórico: Definir e explicar os conceitos de variáveis, operadores e estruturas condicionais.
   - Código-fonte comentado: Explicação de cada parte do código desenvolvido, destacando o uso das estruturas condicionais.