Fundamentos do Desenvolvimento Android

Prof. Thiago Vieira de Aguiar

Roteiro

- **Introdução a Kotlin**
  - Collections
  - Null Safety

Collections

Collections: Overview

Uma Collection é uma estrutura que contém um certo número de elementos (que pode ser zero) de um mesmo tipo.

Exemplo:

- List
- Set
- Map

Collections: Tipos

- **Read-only:** provê operações para *acessar* elementos.
- **Mutable:** estende a anterior, com operações de *adição*, *remoção* e *atualização* de elementos.

Collections: Collection<T>

Collection<T> é a implementação básica de uma coleção. Desta se estendem as outras implementações como as anteriores.

```
fun printAll(
    strings: Collection<String>
        ) {
        for(s in strings) print("$s ")
        println()
    }
```

Collections: List<T>

List<T> guarda elementos em uma ordem específica e provê um índice, começando do zero, de acesso.

Elementos em uma *List* (incluindo nulls) podem ser duplicados.

```
val numbers = listOf(
    "one", "two",
    "three", "four")
println("Number of elements: ${numbers.size}")
println("Third element: ${numbers.get(2)}")
println("Fourth element: ${numbers[3]}")
```

Collections: MutableList<T>

MutableList<T> é uma lista com operações de escrita como add e remove, sobre uma posição específica de um elemento.

```
val numbers = mutableListOf(
    1, 2, 3, 4
)
numbers.add(5)
numbers.removeAt(1)
numbers[0] = 0
numbers.shuffle()
```

Collections: Set<T>

Set<T> guarda elementos e provê um índice, começando do zero, de acesso.

Elementos em uma *Set* (incluindo null) não podem ser duplicados.

```
val numbers = setOf(1, 2, 3, 4)
println("Number of elements: ${numbers.size}")
if (numbers.contains(1))
    println("1 is in the set")
```

Collections: MutableSet<T>

MutableSet<T> é uma Set com operações da interface MutableCollection.

```
val numbers = mutableSetOf(
    1, 2, 3, 4)
numbers.remove(1)
numbers.forEach{
    println(it)
}
println("Number of elements: ${numbers.size}")
if (numbers.contains(1))
    println("1 is in the set")
```

Collections: Map<K, V>

Map<K,V> armazena pares de chave-valor, sendo única cada chave, porém contendo valores que podem se repetir.

Pode-se acessar um valor, buscando por sua chave.

```
val numbersMap = mapOf(
    "key1" to 1,
    "key2" to 2,
    "key3" to 3,
    "key4" to 1)
println("${numbersMap[Key1]}")
```

Collections: MutableMap<K, V>

MutableMap<K,V> é um Map com operações de escrita.

```
val numbersMap = mutableMapOf(
    "one" to 1, "two" to 2)
numbersMap.put("three", 3)
numbersMap["one"] = 11
```

Collections: Filtering

A função de filtragem básica é filter()

Ela retorna uma coleção de elementos que atendam um requisito passado.

```
val numbers =
    listOf("one", "two", "three", "four")
val longerThan3 =
    numbers.filter { it.length > 3 }
println(longerThan3)
```

```
val numbersMap = mapOf(
    "key1" to 1, "key2" to 2,
    "key3" to 3, "key11" to 11)
val filteredMap = numbersMap.filter {
    (key, value) -> key.endsWith("1") && value > 10
}
println(filteredMap)
```

Collections: Testing

any: Retorna true se ao menos um elemento atende a condição.

none: Retorna true se nenhum elemento atende a condição.

all: Retorna true se todos os elementos atendem a condição.

```
val numbers = listOf(
    "one", "two", "three", "four")

println(numbers.any { it.endsWith("e") })
println(numbers.none { it.endsWith("a") })
println(numbers.all { it.endsWith("e") })
```

Collections: Finding element positions

In any lists, you can find the position of an element using the functions indexOf() and lastIndexOf().

```
val numbers = listOf(1, 2, 3, 4, 2, 5)
println(numbers.indexOf(2))
println(numbers.lastIndexOf(2))
```

Collections: Adding

To add elements to a specific position in a list, use add() and addAll() providing the position for element insertion as an additional argument. All elements that come after the position shift to the right.

```
val numbers = mutableListOf(
    "one", "five", "six")
numbers.add(1, "two")
numbers.addAll(2,
        listOf("three", "four"))
println(numbers)
```

Collections: Removing

To remove an element at a specific position from a list, use the removeAt() function providing the position as an argument. All indices of elements that come after the element being removed will decrease by one.

```
val numbers = mutableListOf(1, 2, 3, 4, 3)
numbers.removeAt(1)
println(numbers)
```

Null Safety

Null Safety

```
var a: String = "abc"
a = null // compilation error
```

Null Safety

```
var b: String? = "abc"
b = null // ok
print(b)
```

Null Safety: Checking

```
val b: String? = "Kotlin"
if (b != null && b.length > 0) {
    print("String of length ${b.length}")
} else {
    print("Empty string")
}
```

Null Safety: Safe Call

This returns b.length if b is not null, and null otherwise.

```
val a = "Kotlin"
val b: String? = null
println(b?.length)
println(a?.length) // Unnecessary
```

Null Safety: not-null assertion

The not-null assertion operator (!!) converts any value to a non-null type and throws an exception if the value is null.

Dúvidas?

Referências

- https://kotlinlang.org/
- https://kotlinlang.org/docs/reference/null-safety.html