Mobile Application Development

Produced by

David Drohan (ddrohan@wit.ie)
Dr. Siobhan Drohan (sdrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology
http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Introducing Kotlin Syntax - Part 2.2

Agenda for Part 2

- ❑ Writing Classes (properties and fields)
- ❑ Data Classes (just for data)
- ❑ Collections: Arrays and Collections
- ❑ Collections: *in* operator and lambdas
- ❑ Arguments (default and named)

Agenda for Part 2

- ❑ Writing Classes (properties and fields)
- ❑ Data Classes (just for data)
- ❑ Collections: Arrays and Collections
- ❑ Collections: *in* operator and lambdas
- ❑ Arguments (default and named)

Collections

Arrays and Collections

Arrays (using arrayOf)

- Arrays in Kotlin can be created using arrayOf() or the Array() constructor.

```
fun main(args: Array<String>) {

  val myArray = arrayOf(4, 5, 6, 7)
  println(myArray.asList())
  print(myArray[2])

}
```

Console

```
<terminated> Config - Main.kt [Java Ap
[4, 5, 6, 7]
6
```

Arrays (using arrayOf)

- You can create an array of mixed types (different from Java)

```
fun main(args: Array<String>) {

    val myArray = arrayOf(4, 5, 6, 7, "mixed", "types", "allowed")
   print(myArray.asList())
}
```

Console

```
<terminated> Config - Main.kt [Java Application] C:\Program
[4, 5, 6, 7, mixed, types, allowed]
```

```
fun main(args: Array<String>) {

  val intArray1     = intArrayOf(4, 5, 6, 7)
  val intArray2     = arrayOf<Int>(4, 5, 6, 7)
  val charArray     = charArrayOf('a', 'b', 'c', 'd')
  val booleanArray = booleanArrayOf(true, false, true)

  val mixedArray1 = intArrayOf(4, 5, 6, 7, "will","not","compile")
  val mixedArray2 = arrayOf<Int>(4, 5, 6, 7,"will","not","compile")
}
```

Arrays (using arrayOfNulls)

- Or an array of **nulls**, in this case, to hold **Ints**

```
fun main(args: Array<String>) {

    val nullArray = arrayOfNulls<Int>(5);
    println (nullArray.asList())

}
```

```
Console
<terminated> Config - Main.kt [Java Application] C:\Program Files\Java\
[null, null, null, null, null]
```

Arrays (using constructor)

- The Array() constructor requires a **size** and a **lambda function**.

```
fun main(args: Array<String>) {

    val intArray = Array(6, { i -> i * 2 })
    print (intArray.asList())

}
```

Value to be inserted into the index

Index of the array element

Console

```
<terminated> Config - Main.kt [Java
[0, 2, 4, 6, 8, 10]
```

Collections

- Unlike many languages, Kotlin distinguishes between mutable and immutable collections (lists, sets, maps, etc).

- Precise control over exactly when collections can be edited is useful for eliminating bugs, and for designing good APIs.

Collections – mutable Vs immutable

- The Kotlin List<out T> type is an interface that provides read-only operations like size, get and so on.
- Like in Java, it inherits from Collection<T> and that in turn inherits from Iterable<T>.
- Methods that change the list are added by the MutableList<T> interface.
- This pattern holds also for Set<out T>/MutableSet<T> and Map<K, out V>/MutableMap<K, V>.

Collections – mutable List

```kotlin
fun main(args: Array<String>) {

  // Create a mutable list (MutableList).
  val fruit = mutableListOf("Banana", "Kiwifruit", "Mango", "Apple")
  println(fruit)

  // Add a element to the list.
  fruit.add("Pear")
  println(fruit)

  // Change an element in the list.
  fruit[1] = "Orange"
  println(fruit)

  // Remove a existing element from the list.
  fruit.removeAt(2)
  println(fruit)
}
```

Console

```
<terminated> Config - Main.kt [Java Application] C:\Program
[Banana, Kiwifruit, Mango, Apple]
[Banana, Kiwifruit, Mango, Apple, Pear]
[Banana, Orange, Mango, Apple, Pear]
[Banana, Orange, Apple, Pear]
```

Collections – immutable List – example 1

```
fun main(args: Array<String>) {

    val numbers: MutableList<Int> = mutableListOf(1, 2, 3)
    val readOnlyView: List<Int> = numbers

    println(numbers)          // prints "[1, 2, 3]"
    numbers.add(4)

    println(readOnlyView)     // prints "[1, 2, 3, 4]"
    readOnlyView.clear()      // -> does not compile

}
```

Collections – immutable List – example 2

```
class Person( _firstName: String = "UNKNOWN",
              _lastName: String = "UNKNOWN") {

    private val _items = mutableListOf<String>("1", "2", "3")
    val items: List<String> get() = _items.toList()
}
```

custom function get() 'linking' _items with items

items returns a snapshot of a collection at a particular point in time (that's guaranteed to not change) as toList() just duplicates _items.

Console

```
<terminated> Config - Main.kt [Java Applica
[1, 2, 3]
```

```
fun main(args: Array<String>) {

    val person = Person()
    println(person.items)
    //person.items.clear()  //doesn't compile
}
```

Collections – Set and hashSet

```
fun main(args: Array<String>) {

    // mutatble set
    val mutableSet : MutableSet<Int> = mutableSetOf(1,2,3)
    println(mutableSet)
    mutableSet.add(4)
    println(mutableSet)

    // immutatble set
    val immutableSet : Set<Int> = setOf(9,8,7)
    println(immutableSet)
    //immutableSet.add(6)  //won't compile

    //note: ignores duplicate items
    val strings = hashSetOf("a", "b", "c", "c")
    println("Size: ${strings.size}, Contents: " + strings)
    strings.add("d")
    println("Size: ${strings.size}, Contents: " + strings)
}
```

Console

```
<terminated> Config - Main.kt [Java Application] C:\Pr
[1, 2, 3]
[1, 2, 3, 4]
[9, 8, 7]
Size: 3, Contents: [a, b, c]
Size: 4, Contents: [a, b, c, d]
```

Collections – Map and hashMap

```
fun main(args: Array<String>) {

  // mutatble map
  val mutableMap = mutableMapOf("W" to "Watreford", "C" to "Cork")
  println(mutableMap)
  mutableMap.put("D", "Dublin")
  println(mutableMap)
  mutableMap["W"] = "Waterford"
  println(mutableMap)

  // immutatble map
  val immutableMap : Map<Int, String> = mapOf(1 to "One", 2 to "Two")
  println(immutableMap)
  //immutableMap.put(3, "Three")  //won't compile
}
```

Console

```
<terminated> Config - Main.kt [Java Applicati
{W=Watreford, C=Cork}
{W=Watreford, C=Cork, D=Dublin}
{W=Waterford, C=Cork, D=Dublin}
{1=One, 2=Two}
```

Some additional sources for exploration:

Inheritance	https://www.programiz.com/kotlin-programming/inheritance
Interfaces	https://www.programiz.com/kotlin-programming/interfaces
Collections	https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/index.html
Try examples online	https://try.kotlinlang.org/#/Examples/Hello,%20world!/Simplest%20version/Simplest%20version.kt
Encapsulation & Polymorphism	https://medium.com/@napperley/kotlin-tutorial-12-encapsulation-and-polymorphism-6e5a150f25e1
Spek (testing)	https://objectpartners.com/2016/02/23/an-introduction-to-kotlin/ https://github.com/mike-plummer/KotlinCalendar

References

Sources:

http://kotlinlang.org/docs/reference/basic-syntax.html
http://petersommerhoff.com/dev/kotlin/kotlin-for-java-devs/
https://www.programiz.com/kotlin-programming
https://www.baeldung.com/kotlin-lambda-expressions
https://www.programiz.com/kotlin-programming/lambdas
https://medium.com/@napperley/kotlin-tutorial-5-basic-collections-3f114996692b


Thanks.