Programming with Android:
Kotlin for Android

Federico Montori

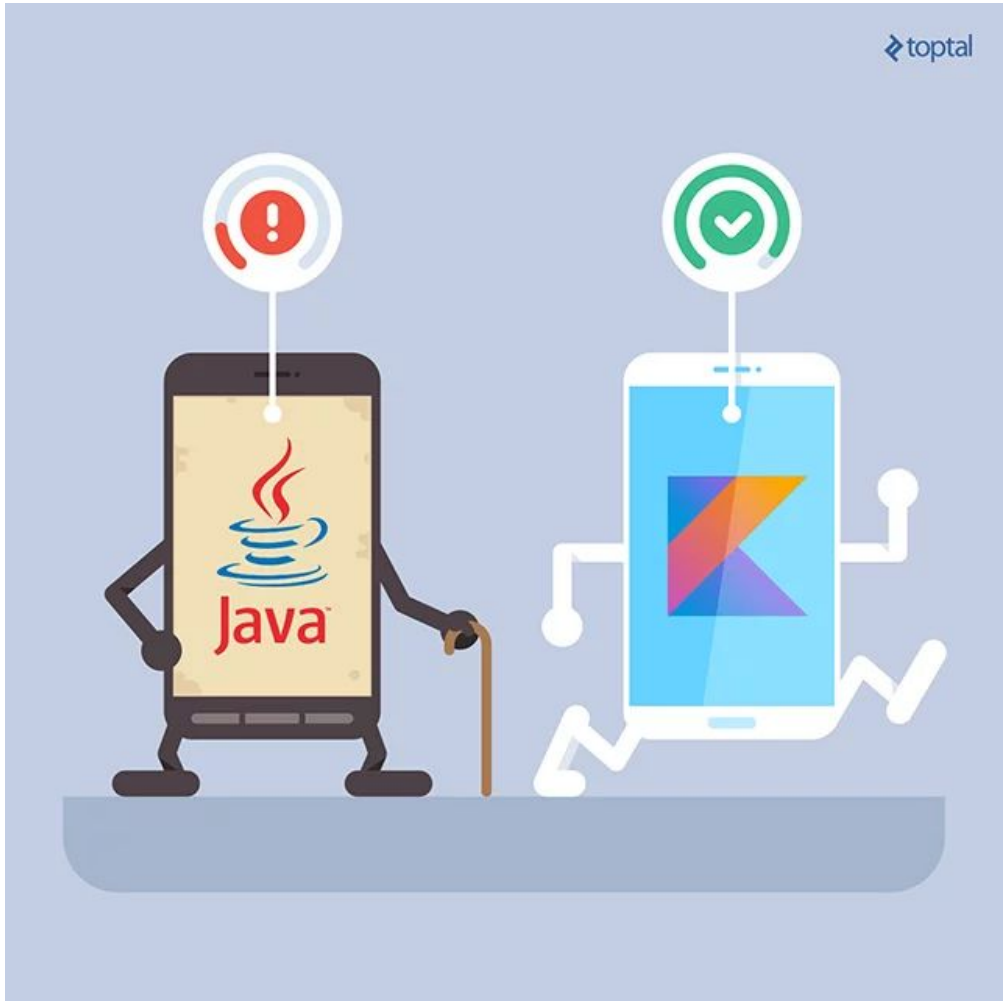
Dipartimento di Scienze dell'Informazione

Università di Bologna

Outline

- Android and Kotlin
- Getting started with Kotlin
- **Kotlin Tutorial:** Fundamentals
- **Kotlin Tutorial:** Null Safety
- **Kotlin Tutorial:** Lambdas
- **Kotlin Tutorial:** Classes
- Java and Kotlin under comparison

Android: Java and Kotlin



Why Java?

It's been the official language for years and most supported until last year.

As for now, it's not the most used, Kotlin took over this year, however since we know Java we can focus on the Mobile Architecture.

Android: Kotlin

It is the **official programming language for Native Android** since 2019

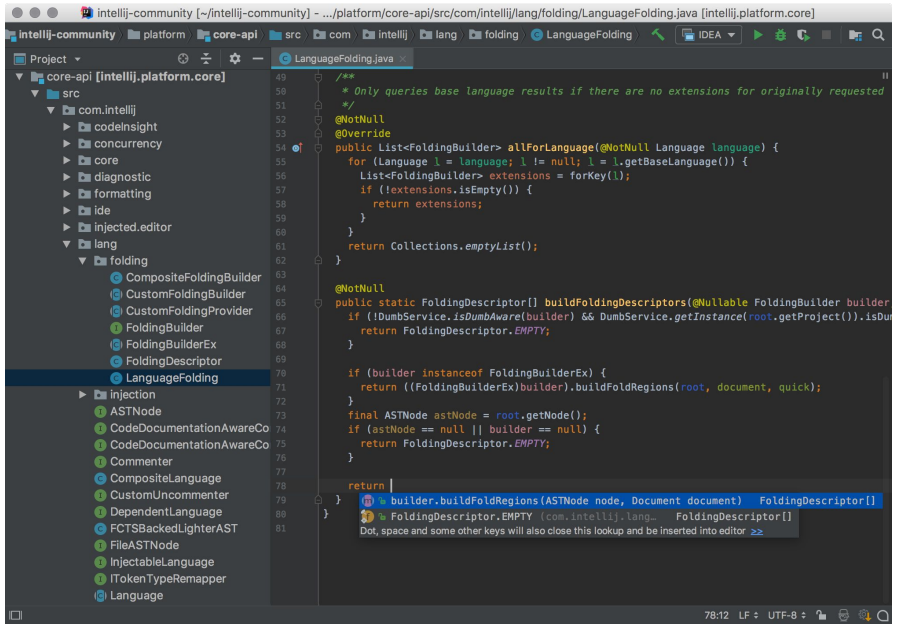
- Announced by JetBrains in 2011
- New language for the JVM
- Open source since 2012 under Apache 2 License
- Named after Kotlin Island
  - FYI Java is an island too

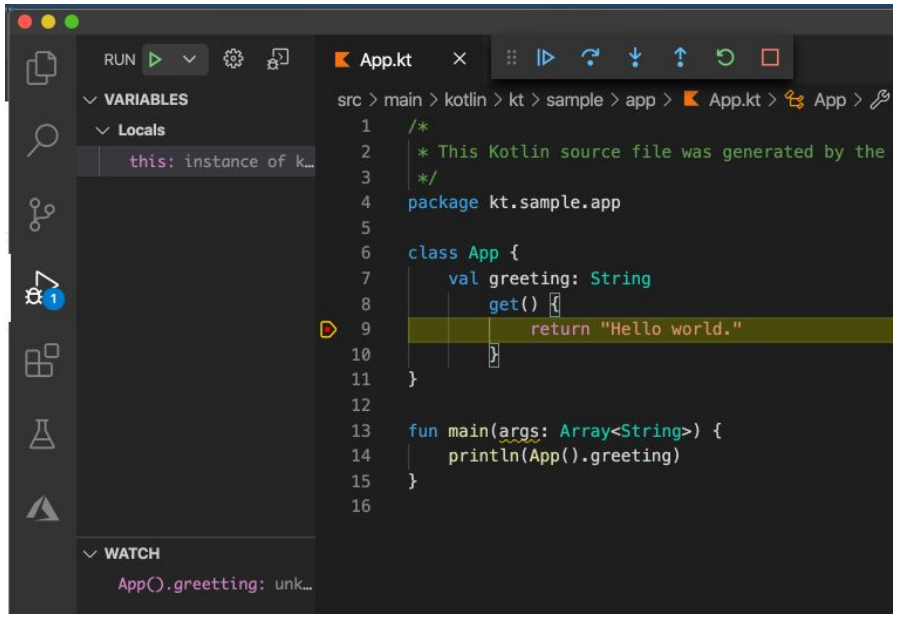
Kotlin: Kotlin General Features

- It is a **Type Inference** language (like Python)
  - Still, it is **statically typed**
- It is **Cross-Platform**
- It compiles to **Java Bytecode**
  - Fully interoperable with Java
  - You can write easily mixed code projects
  - It can also compile to Javascript and other stuff

Kotlin Tutorial: Getting started with Kotlin

Kotlin is Cross-Platform →like Java, it is not bound to Android





Intellij IDEA (supported natively)

Visual Studio Code

Basically the brother of Android Studio…

Kotlin Tutorial: Variables and Types

Declaration of variables and types

```
var x: Int = 42          // Declaration of a variable with type Int
var x = 42               // Declaration of a variable with inferred type Int
val x = 42               // Declaration of a constant with inferred type Int
```

Type inference does not mean that types are dynamic (like in Python…)

```
var x = 42
x = 'c'                  // This will give an error
```

Disclaimer: this is an accelerated tutorial

Complete official guide: https://kotlinlang.org/docs/home.html

Kotlin Tutorial: Variables and Types

Basic types:

- Int
- Long
- Short
- Byte
- Float
- Double
- Boolean
- Char
- String

Can always specify them, or:

```
var x = 42
var x = 42L


var x = 42.42f
var x = 42.42
var x = true
var x = ‘f’
var x = “fortytwo”
```

Kotlin Tutorial: Operators

Operations in Kotlin are quite straightforward…

- Arithmetic Operators
  - + - * / %
- Logical Operators
  - && || !
- Comparison Operators
  - < > == >= <= !=

Kotlin Tutorial: Strings and Prints

Like some other imperative languages, the access point is the **main** function.

```
// Enhanced Hello World Example
fun main() {
    val nickname: String = “stradivarius”
    println(“Hello world, my name is $nickname”)
}
```

Kotlin Tutorial: Selection Construct

The IFTE construct is straightforward too…

```
if ( condition ) {
    // Then Clause
} else {
    // Else Clause
}
```

There is a contract syntax for assignments

```
var y = if (x == 42) 1 else 0
```

Kotlin Tutorial: Selection Construct

The case construct is as follows

```
when ( x ) {
    in 0..21 -> println("One line clause")
    in 22..42 -> println {
        println("Multiple line clause")
    }
    else -> println("Default clause")
}
```

With the double dot (..) you can specify ranges, which originate Lists (see later).

Kotlin Tutorial: Arrays and Lists

```
val arr: Array<Int> = intArrayOf(1, 2, 3)          // [1,2,3]
println(arr[0])
```

Arrays are a class and can be instantiated in several ways (they also have their subtypes):

```
// Array of int of size 5 with values [0, 0, 0, 0, 0]
val arr = IntArray(5)

// Array of int of size 5 with values [42, 42, 42, 42, 42]
val arr = IntArray(5) { 42 }

// Array of int of size 5 with values [0, 1, 2, 3, 4] (lambda, you'll see…)
var arr = IntArray(5) { it * 1 }
```

Kotlin Tutorial: Arrays and Lists

Lists are similar to Java ArrayLists and can be “constants” or “variables”.

```
// Immutable List
val myList = listOf<String>(“one”, “two”, “three”)
println(myList[0])

// Mutable List (referenced by a val because it is the pointer)
val myMutableList = mutableListOf<String>(“one”, “two”, “three”)
myMutableList.add(“four”)
```

Kotlin Tutorial: Loops

The iteration constructs are straightforward too…

```
// While loop
var counter = 0
while (counter < myMutableList.size) {
    println(myMutableList[counter])
    counter++
}

// For loop
for(item in myListMutable)          // Here we can use ranges as well
    println(item)
```

Kotlin Tutorial: Null Safety

One of the major advantages of Kotlin is the **Null Safety**

➔ The program does not crash because of null values (remember the annoying Java **NullPointerException**)

- Basically types are non-nullable, in fact variables are either:
  - Initialized
  - Explicitly null, but they throw error at compile time
- Variables that can be null are *Nullable* but calling them is safe

let's see how…

Kotlin Tutorial: Null Safety

Non nullable types

```
var s: String = “Hello”   // Regular initialization means non-null by default
s = null                  // compilation error
```

Nullable types

```
var s: String? = “Hello”// Nullable initialization means it can be null
s = null                // this is ok: e.g. if you print it, it will print “null”
```

Null safety

```
val l = s.length                    // Compiler error: “s can be null”
val l = s?.length                   // If s is null then l is null (if nullable)
val l = if (s != null) s.length else -1   // Custom workaround
```

Kotlin Tutorial: Null Safety

This is true even for more complex scenarios, for instance:

```
val name: String? = department?.head?.getName()
name? = department.head.getName()
```

If anything in here is null, then the function is not called

You really want it to be not null:

```
val l = s!!.length                    // Casts s to non nullable, can throw exception
```

The “Elvis” operator

```
val l = s?.length ?: -1               // -1 is the default value for l if s is null
```

Kotlin Tutorial: Functions

Ordinary functions (they support the default value)

```
fun isEven(number: Int = 0): Boolean {  // number is set to 0 if not passed
    return number % 2 == 0
}
isEven(14)
```

Extension functions

```
fun Int.isEven(): Boolean {                    // Extend the class Int
    return this % 2 == 0
}
14.isEven()
```

Kotlin Tutorial: Higher Order Functions

Higher order functions take functions as inputs

```
fun List<String>.customCount(function: (String) -> Boolean): Int {
        var counter = 0
        for (str in this) {
                if (function(str))
                        counter++
        }
        return counter
}                                   // Function that counts members in a List of strings that respect a certain condition
```

They might as well take any type in (usually called “generics”)

```
fun <T> List<T>.customCountAllTypes(function: (T) -> Boolean): Int {
        var counter = 0
        for (anything in this) {
                if (function(anything))
                        counter++
        }
        return counter
}                                   // Function that counts members in a List of any type that respect a certain condition
```

Kotlin Tutorial: Lambdas

Lambdas are undeclared functions that are passed directly as they are and used once.

➔ Added to Java as well (sometimes we use it with onClickListener…)

Let us use the previous higher order functions…

```
val myList = listOf<String>("one", "two", "three")

val x: Int = myList.customCount { str -> str.length == 3 }

val x: Int = myList.customCountAllTypes { str -> str.length == 3 }
```

Kotlin Tutorial: Classes

Classes are pretty much like in Java, however they typically have a primary constructor:

```
class Animal (                          // Constructor is within round brackets
    val name: String,
    val legCount: Int = 4               // Default value if not passed
) {
    var sound: String = “Hey”           // Property not initialized by the constructor

    init {
        println(“Hello I am a $name”)   // Function executed at instantiation time
    }
}
val dog = Animal(“dog”)                 // Instantiation of a class into an object
val duck = Animal(“duck”, 2)
```

Kotlin Tutorial: Classes

Properties have default accessors (setters, getters…)

you can define custom ones or make it private…

```
// Equivalent notation
var sound: String = “Hey”
    get() = field
    set(value) { field = value }          // Keyword field refers to the property

// Custom notation
var sound: String = “Hey”
    get() = this.name
    private set                           // Setter is private

val dog = Animal(“dog”)
dog.sound                                 //  Will access the getter, not the property
```

Kotlin Tutorial: Classes

You can obviously subclass that if the original class is **open**

```
class Dog: Animal(“dog”) {
    fun bark() {
        println(“WOOF”)
    }
}

class Duck: Animal(“duck”, 2) {
    fun quack() {
        println(“QUACK”)
    }
}
```

Kotlin Tutorial: Classes

Let us make that abstract

```
abstract class AbstractAnimal (
        val name: String,
        val legCount: Int = 4
) {
        abstract fun makeSound()
}
```

Then you'll have to implement the abstract method

```
class Cat: AbstractAnimal("cat") {
        override fun makeSound() {
                println("MEOW")
        }
}
```

Kotlin Tutorial: Scope Functions

Scope functions are used to simplify multiple interaction with the same object:

```
val snake = Animal("snake")                    // Without “apply”
snake.legCount = 0
snake.sound = "Hiss"
```

```
val snake = Animal("snake").apply {            // With “apply”
    legCount = 0
    sound = "Hiss"
}
```

There are other Scope Funtions: **let**, **with**, **run** and **also**

Read the full doc here: https://kotlinlang.org/docs/scope-functions.html

Kotlin Tutorial: Classes

Finally, you can create an anonymous class, if used only once:

```
val bear = object: AbstractAnimal("bear") {
    override fun makeSound() {
        println("GROWL")
    }
}
```

This concludes our crash tutorial on Kotlin…

Now let us make a recap on the whys and why nots…

Kotlin and Java: Differences

We've seen the similarities between kotlin and Java, what about the differences?

- Explicit types
- Strictly OOP
- Not Null Safe
- Explicit set & get

- Type inference
- Not necessarily OOP
- Null Safe
- Implicit set & get
- + Extension functions
- + Scope Functions
- + Lambdas
- + Implicit Casting
- + Structured Concurrency
  - Coroutines (TBC)

Kotlin and Android

How to set up an Android project in Kotlin?

Literally in the same way it is done for Java!

- Still uses XML resources
- Everything still applies to what we have seen so far:
  - Resources
  - Activity Lifecycle
  - Fragments
  - Intents
  - Views
- Only thing that changes is the syntax…

Kotlin and Android: Let's code!

We will see an example of an application that touches the main topics we have seen so far.