Slide 1

Intro to Kotlin (Kotlin vs Java) Testing your First Android Kotlin application
Week 1_1 2025 W Dr. Volodymyr Voytenko

Slide 2

Kotlin ?
Kotlin is a programming language from JetBrains, from 2011 If you're familiar with Java but not Kotlin, that's fine – they share many similarities! Until 2017 Android app development required the use of Java Kotlin was officially recommended for Android development by Google on May 17, 2017, during the Google I/O conference. This announcement marked Kotlin as a first-class language for Android development. Kotlin was favored because it offers modern features like null safety, concise syntax, and better interoperability with existing Java code, leading to fewer errors and increased developer productivity.

Slide 3

Benefits of Kotlin
Kotlin is Open Source Kotlin compiles to JVM bytecode or JavaScript Kotlin programs can use all existing Java Frameworks and Libraries Automatic conversion of Java to Kotlin Kotlin’s null-safety Kotlin can be learned easily

Slide 4

Java vs Kotlin

Slide 5

Kotlin VS Java: variable declaration
Kotlin is a statically typed language. This means that a variable can only be assigned to objects of one fixed type, the type of the variable:
var m : Int = 17 m = 18 // ok m = "seventeen" // Error, wrong type! m = 18.0 // Error, wrong type!

Slide 6

Kotlin : constants vs variables
Use val to declare a constant, and var keywords to declare a variable.
val m = 17 m = 18 // error!
A var variable, on the other hand, can change its value as often as you want:
var n = 17 n = 18 // ok

Slide 7

Kotlin : basic data types
To convert a number from one type to another, you have to explicitly invoke function:
Int - integer in the range −231 to 231−1 Long - larger than the maximal number allowed by the Int type. Boolean - either true or false Double Double - floating-point representation of a real number. Char - single character String - sequence of characters such as "Hello World".

Slide 8

Kotlin : arrays
There are two main ways to create an array: using function arrayOf() or the constructor Array().
To enforce that all the array values have the same type, e.g. Int, we declare a type by calling arrayOf() or intArrayOf():
Now let's see how to create an array with Array(). The constructor of this class requires a size and a lambda function. It is a simple, inline way of declaring an anonymous function. In this case, the job of the lambda function is to initialize the array with elements:
In the code above, we passed 5 as the size of the array in the first argument. The second argument takes in a lambda function, which takes the index of the array element and then returns the value to be inserted at that index in the array. So in the example above, we created an array with elements 0, 2, 4, 6, and 8.

Slide 9

Kotlin functions : declarations and usage

Slide 10

Control structures in Kotlin

Slide 11

Control Flow: if
Unlike Java (and other many programming languages), if can be used an expression in Kotlin; it returns a value.
Like in Java (traditional if)
The else branch is mandatory when using if as an expression.
The curly braces are optional if the body of if has only one statement:

Slide 12

Control Flow: when
The when construct in Kotlin can be thought of as a replacement for Java switch Statement. It evaluates a section of code among many alternatives.

Slide 13

Control Flow: for
There is no traditional for loop in Kotlin unlike Java and other languages. The for loop in Kotlin iterates through anything that provides an iterator – to iterate through ranges, arrays, maps and so on (anything that provides an iterator). The syntax of for loop in Kotlin is:
Example: Iterate Through a Range
Iterating Through an Array

Slide 14

Control Flow: while, do while
Loop is used in programming to repeat a specific block of code until certain condition is met (test expression is false).

Slide 15

Kotlin Null Safety

Slide 16

Declare variable with nullable property
In Kotlin, the type system distinguishes between null values (nullable references) and those that can not be null (non-null references). In example below, a normal property can’t hold a null value and will show a compile error.
We can add a ”?” after the data type of that property which declares that variable as a nullable property
var a: String = "abc" a = null // compilation error!
To allow nulls, we can declare a variable as nullable string, written String?:
var a: String? = "abc” // added “?” mark after type a = null // ok print(a)
Null can not be a value of a non-null type String

Slide 17

Checking for null in conditions
How to deal with a compile error when using variable with null property?
var b: String? = "abc" b = null // ok
But if you want to access the same property on b, that would not be safe, and the compiler reports an error again:
val l = b.length // compliation error: variable 'b' can be null Only safe (?.) or non-null asserted (!!.) calls are allowed on a nullable receiver of type String?
val l = b?.length // OK if the property is not null or returns null if that property is null without NPE exception
Solution 1: add ? after variable to make “safe call”

Slide 18

All way for checking for null in conditions
How to deal with null property for var b: String? = "abc” ■ val l = b.length
Method 1) Explicit Null Check – old pattern
val l = if (b != null) b.length else -1
Method 2) Safe Calls (?.)
Another way of using a nullable property is safe call operator ?. This calls the method if the property is not null or returns null if that property is null without throwing an NPE (null pointer exception).
We can explicitly check if b is null, and handle the two options separately. This is the old pattern that we use in every other language.
val l = b?.length
Method 4) The !! Operator
This operator is used to explicitly tell the compiler that the property is not null and if it’s null, please throw a null pointer exception (NPE).
val l = b!!.length
So, this will return a non-null value of b (e.g., a String in our example) or throw an NPE if b is null
Method 3) Elvis Operator (aka Java conditional operator “?”)
When we have a nullable reference b, we can say "if b is not null, use it, otherwise use some non-null value, f.e., -1:
val l = b?.length ?: -1
1
2
3
4
Still have the questions? Read the guide: https://kotlinlang.org/docs/reference/null-safety.html

Slide 19

Kotlin classes

Slide 20

Kotlin VS Java : user classes (data classes)
Data Classes – In Kotlin there are Data Classes which leads to autogenerating of boilerplate functions like toString(), getters/setters, equals(), hashCode(), and much more!

Slide 21

Kotlin vs. Java, cont.
Data class and its copy
… Java class Book
Kotlin data class Book

Slide 22

Getting Started with Android App development Using Kotlin

Slide 23

First “Hello Android” app
Proceed through the Android Studio wizard, with “New project”, by selecting “Empty Activity”, as development tool, setup up your new Android project as shown, and press “Finish”:

Slide 24

First “Hello Android” app
When complete, take a look at the Kotlin source code for your Activity in the MainActivity.kt file. By default, Kotlin files are stored alongside Java source files and can be found in the kotlin+java folder when using the Android project view:

Slide 25

First “Hello Android” app, with Scaffold
package declaration
importing classes
class declaration
function inside the class
the function declaration outside of the class
the function call with 2 parameters
Scaffold refers to a composable layout in Jetpack Compose that provides a structure for building consistent UI screens. It includes slots for common UI elements like app bars, floating action buttons, navigation drawers, and content, making it easier to manage layouts and maintain a cohesive design.

Slide 26

The same First “Hello Android” ap, simplified without Scaffold (basic version)
package declaration
importing classes
class declaration
function inside the class
the function declaration outside of the class
the function call with 1 parameter
Simplified version of first “Hello Android” app
You can create simple app without using Scaffold. Scaffold is just a helper for structuring common UI elements, we will use it next weeks!

Slide 27

What to do next ?
Complete the Android Studio installation and set up a device emulator. To test Android Studio, perform the following: Create your first Android app, "Hello, Android." Modify the app to function without using Scaffold. Were both apps successfully runnable? What differences did you observe? If you encounter any issues running Android Studio, consult your instructor for assistance.

Slide 28

REFERENCES
https://kotlinlang.org/docs/reference https://dac.digital/kotlin-vs-java/
https://www.xenonstack.com/blog/kotlin-andriod/ https://code.tutsplus.com/tutorials/kotlin-from-scratch
https://www.programiz.com/kotlin-programming/while-loop