NARESHi technologies
An ISO 9001 : 2008 Certified Company

Android App Development Using Kotlin

Batch 26

Recorded Sessions (Only Demo Sessions)

- https://youtu.be/WzuSi5IGf8Q
- https://youtu.be/bLQ1WNMxRwk
- https://youtu.be/Bv1ye04scso
- https://youtu.be/ySnwnAM-WA0
-

Take a look at the syllabus:

✪ Introduction to Android Development

- What is Android?
- Evolution of Android & Key Features
- Android Market Trends (Kotlin, Jetpack, Compose)
- Android App Development Workflow
- Android Ecosystem (Phones, Tablets, Wearables, TV)

🧭 Setting Up Development Environment

- Installing Android Studio & SDK
- Configuring Gradle & Dependencies
- Project Structure in Android Studio
- Running on Emulator & Physical Device

🔷 Kotlin Programming for Android

- Kotlin Basics (Variables, Data Types, Operators)
- Control Flow & Functions (if-else, loops, when)
- OOP in Kotlin (Classes, Objects, Inheritance)
- Kotlin Extensions & Higher-Order Functions
- Null Safety & Safe Calls
- Kotlin Coroutines & Flow (Replacing AsyncTask)

NARESHO technologies
An ISO 9001 : 2008 Certified Company

📌 **UI Development (Jetpack Compose + XML)**

XML-Based UI (For Legacy & Hybrid Projects)

- Basic Views & Layouts (LinearLayout, ConstraintLayout, ScrollView)

Jetpack Compose (Modern UI)

- Introduction to Jetpack Compose
- Composables & UI Elements
- Text, Buttons, Image, LazyColumn, LazyRow
- State Management (Remember, MutableState)
- Layouts & Theming (Material 3, Dark Mode)
- Navigation in Compose
- Working with Lists & Adapters in Compose
- Animations & UI Interactions

🧭 **Jetpack Architecture Components**

- ViewModel – Managing UI Data
- LiveData & StateFlow – Observing Data Changes
- Room Database – Local Database with ORM
- WorkManager – Background Tasks & Scheduling
- Navigation Component – Handling Screens
- Services (Foreground & Background Services)
- Broadcast Receiver
- Content Provider

🌐 **Networking & APIs**

- Retrofit with Coroutines (GET, POST, PUT, DELETE)
- Parsing JSON Responses with Moshi/Gson
- Dependency Injection (Dagger/Hilt)
- REST API Authentication (JWT, OAuth, Firebase Auth)
- Handling Network Errors & Caching
- JSON Parsing

🔗 **Firebase & Cloud Integration**

- Firebase Authentication (Google, Email, OTP Login)
- Firestore Database (NoSQL Cloud Storage)
- Firebase Push Notifications (FCM)
- Crashlytics & Performance Monitoring

🎥 **Media & Device Features**

- Image & Video Handling with CameraX
- Audio & Video Playback (ExoPlayer)
- Permissions & Security Best Practices

📍 **Location & Google APIs**

- Google Maps Integration

🔒 **Security & App Performance**

- Best Practices for Secure Android Apps
- Jetpack Security (Encrypted SharedPreferences, Biometric Auth)
- App Optimization & Performance Tuning
- Dynamic Permissions Handling & User Consent

📺 **User Interaction & Notifications**

- Dialogs (AlertDialog, Custom Dialogs, BottomSheets)
- Notifications (Push, Local, Advanced Notifications)

📦 **Publishing & Play Store Deployment**

- Building a Signed APK/AAB
- Google Play Console Walkthrough
- App Store Listing, Policies & Monetization
- In-App Purchases & Ads (AdMob, Subscription Model)

Duration, Timings & Communication Channels

Duration - 45 (50-60) lectures (Each lecture is 1 day)
Timings - 7:30 AM - 9:00 AM
Demo sessions - 5 days a week (sat & Sun are holidays)
Actual Course sessions - 6 or 5 days a week
Communication Channel

- Gmail (pavankreddy.t@gmail.com)
- WhatsApp Group (Once the demos are completed)

Prerequisites for the course

- Enthusiasm

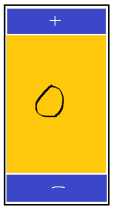
- Programming skills in any one of the programming languages (c, c++, Java, python)
- Computer (desktop/laptop) with good configuration
    - All OS (windows/Mac/linux)
        - 8GB min RAM
        - 20 GB free disk space
        - I3 latest or above
- Need to spare some time every single day to complete the Assignments.

Apr 7, 2025

- Discussed about the syllabus
- Timings, Duration and Communication
- PrerequisitesPresentation 1
- Q & A
- Software Requirements
- Brief history of Android
-
    - Android Studio [download the android studio]
    - Install Android Studio [Assignment]

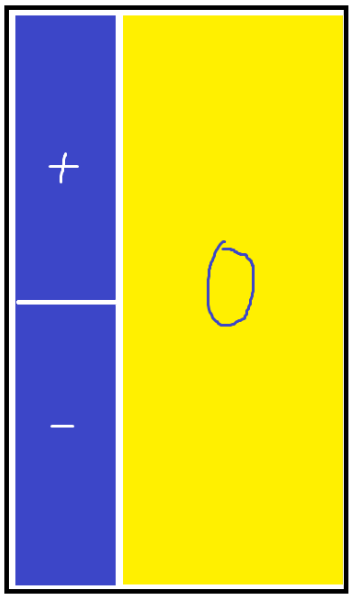
Apr 8, 2025

- Android Platform Architecture
- Creating our First Android Project (Score Keeper)
- Understanding the Project Structure
- Running our app on Emulator and on Physical Device
- What are we going to create for our first project (score Keeper)
-



Apr 9, 2025

- Views, ViewGroups and Layouts - View Hierarchy
- Design our first UI screen
- Assignment 2
    - Develop a design as follows using linear layout (you can also use nesting)

- 

    - After completing the assignment, drop an email with the xml code to pavankreddy.t@gmail.com
- Refer to the code in the project

Apr 10, 2025

- Activity lifecycle
- onSavedInstanceState

Understanding MVC architecture

MVC (Model-View-Controller) is a software architectural pattern that separates an application into three interconnected components to improve modularity and maintainability.

Components in Detail

1. Model (Data Layer)
    a. Represents the data and business logic of the application.
    b. Handles data manipulation, API calls and database operations.
    c. Example: a repository fetching user data from a database or API.
2. View
    a. Represents the UI and is responsible for displaying data to the user
    b. Consists of XML layouts, ACtivities and Fragments
    c. Should be as dumb as possible, meaning it should only display the data received.

3. Controller
    a. Acts as a mediator between the model and the view.
    b. Contains the app's business logic.
    c. In Android, The activity or the fragment often acts as the controller.

How MVC works in Android ?

1. The View (XML, Activity, Fragment) displays the UI and interacts with the user
2. The user triggers an Action (eg., clicking on the button)
3. The Controller (Activity, Fragment) processes the user input and interacts with the Model.
4. The Model fetches or processes data and sends it back to the Controller
5. The controller updates the view with the new Data.

Disadvantages of MVC in Android

- Tightly coupled components: Activities handle both UI and Business logic, leading to poor separation of concerns.
- Difficult to test: Since the controller manages both UI and logic, testing becomes hard.
- Code clutter in Activities: Android Activities act as both controllers and views, making them hard to maintain.

Kotlin Programming language basics for Android app Development

---

Batch 26 - Naresh Technologies

Why Learn Kotlin ?

1. Google Announced kotlin as the official programming language for android development.
2. Over 50% of professional android developers use kotlin as their primary programming language. only 30% of the developers still use java. All kotlin users (Developers) said that kotlin makes them more productive.
3. Benefits
    1. Less code combined with greater readability
    2. Fewer common errors
    3. Kotlin offers a great set of jetpack libraries - extensive support is also there.
    4. Interoperability with java.
    5. Matured language and the environment is also matured.
    6. Easy to pick up
    7. Big community.

Official Kotlin Docs

How to run kotlin programs ?

- use online compiler kotlin play
- Android studio
- Intellij IDEA
- You can also integrate kotlin in visual studio code.

First Kotlin Program

```
package com.nareshtech.scorekeeper

fun main(){
    print("Hello World!")
}
```

1. All your kotlin files has `.kt` extension
2. Your program execution always begins at `main(...)`
3. `;` statement terminators are option in kotlin.
4. Kotlin is both functional programming language and also object oriented programming language. hence, you can write programs that do not contain classes and objects.

`print(...)` vs `println(...)`

println will add a new line character at the end of the string inside it.

Working with Variables in Kotlin

- Two types of variables in Kotlin

- Immutable variables (The value once assigned cannot be changed)
  - Defined using `val` keyword
    - Ex: val x = 10
    - By which, we are creating a variable called `x` and we are assigning `10` to it. And this value cannot be modified later on in any part of the program.
- Mutable Variabales (The value assigned can be changed at any time during the scope of the variable)
  - Defined using `var` keyword.
  - Ex: var x = 10
  - By which, we are creating a variable called `x` and we are assigning `10` to it. And this value can be modified later on.

Data types in Kotlin

- Kotlin offers `TypeInference` feature. This means, based on the value that is assigned, a variable's data type is inferred (decided).
  - Ex: val x = 10
  - since the value of 10 is integer type, the variable `x` data type would be Int.

How can we identify the data type ?

```
fun main(){
    val x = 10L
    val y = 13.5f
    val z = false

    println(x::class.java.simpleName)
    println(y::class.java.simpleName)
    println(z::class.java.simpleName)
}
```

Kotlin Integers

- Kotlin supports a set of built in types that represent numbers
- For `Integer numbers` there are four types with different sizes and hence the value ranges
  - Byte
  - Short
  - Int
  - Long
- While you initialize a variable with no explicit type specification, the compiler automatically infers the type to the smallest range enough to represent the value starting from `Int`
- If the value exceeds the range of Int, the type will be `Long`

- To append a long value, we need to suffix the value with `L`.

Floating Point Data Types

- Float (Single precision) 32 bits
    - Uses 1 bit for sign, 8 bits for exponent, and 23 bits for fraction (mantissa)
    - can store approximately 7 decimal digits of precision.
    - Ex: 3.1415927f may lose some accuracy beyond 7 digits.
- Double (Double Precision) 64 bits
    - Uses 1 bit for sign, 11 bits for exponent, and 52 bits for the fraction(Mantissa).
    - Example: 3.1415926535389793 - maintains higher accuracy.
- To represent the float value add `F` or `f` as a suffix.

Explicit Type Conversion

- toByte()
- toShort()
- toInt()
- toFloat()
- toDouble()

Functions in Kotlin

- All functions in Kotlin must be defined with a keyword called `fun`
- The return type of the function should be mentioned after the function declaration.

```kotlin
package com.nareshtech.scorekeeper

fun main(){
    println(sum(10,20))
    sum(10,20,30)
    println(square(10))
}

fun sum(a:Int, b:Int):Int{
    return a+b
}
```

```
/*fun sum(a:Int, b:Int, c:Int):Unit{
    println(a+b+c)
    return
}
the same function can be written in single line as follows
*/

fun sum(a:Int, b:Int, c:Int) = println(a+b+c)

// you can also write inline functions
fun square(a:Int):Int = a*a
```

Kotlin's interoperability with Java

```kotlin
package com.nareshtech.scorekeeper

// I want to read values from the output console when the user enters them
// We shall use Java's Util packages Scanner class
import java.util.Scanner

fun main(){
    val s:Scanner = Scanner(System.`in`)
    print("Enter first number: ")
    val a = s.nextInt()
    print("Enter second number: ")
    val b = s.nextInt()
    print("Enter third number: ")
    val c = s.nextInt()
    sum(a,b,c)
}

fun sum(a:Int, b:Int):Int{
    return a+b
}

/*fun sum(a:Int, b:Int, c:Int):Unit{
```

```
    println(a+b+c)
    return
}
the same function can be written in single line as follows
*/

fun sum(a:Int, b:Int, c:Int) = println(sum(a,b)+c)

// you can also write inline functions
fun square(a:Int):Int = a*a
```

Condition Control Structure

```
package com.nareshtech.scorekeeper

import java.util.Scanner

// Decide if a number is even or not.
fun main(){
    println("Enter a value")
    val s:Scanner = Scanner(System.`in`)
    val a:Int = s.nextInt()
    // We will have to check if the entered value is even or not.
    /*if(a%2 == 0){
        println("Even Number")
    }else{
        println("Not an Even Number")
    }*/

    /*if(a%2 == 0) println("Even Number") else println("Not an Even Number")*/
    val result = if(a%2==0) true else false

    if(result) print("Even") else print("Not Even")
}
```

Logical Operators in Kotlin

&& - AND || - OR ! - NOT

Assignment

You are given a certain grade of steel and you are also given three values that indicate the steel's properties - Hardness, tensile strength and carbon content. You are supposed to grade the steel based on the conditions below.

1. Hardness must be greater than 50
2. Tensile Strength must be less than 5600
3. Carbon Content must be greater than or equal to 0.7

Take the values from the user and grade it based on the guidelines below

1. If all the conditions are met, grade is 10
2. If only 1 & 2 are true, the grade is 9
3. If only 2 & 3 are true, the grade is 8
4. If only 1 & 3 are true, the grade is 7
5. If only one condition is true among the 3, the grade is 6
6. If no condition is met, the grade is "Not Fit"

When Expressions

`When` is like a switch in Java & C Programming language. `when` is used when you have multiple branches and when the code looks complex with these multiple branches if used with in an If condition.

```
package com.nareshtech.scorekeeper

import java.util.Scanner
import kotlin.math.*

fun main(){
    println("Enter your choice\n1. Double the value\n2. Square\n3. Square root")
    val s:Scanner = Scanner(System.`in`)
    val option = s.nextInt()

    println("Enter the value")
```

```
    val value = s.nextInt()

    when(option){
        1 -> println(doubleValue(value))

        2-> println(powerOfTwo(value))

        3-> println(squareRoot(value))

        else -> println("Invalid option")
    }
}

fun doubleValue(a:Int) = a*2

fun powerOfTwo(a:Int) = a*a

fun squareRoot(a:Int) = sqrt(a.toDouble())
```

Loop Control Structure in Kotlin

When you want tp repeat a set of statements for some number of times or till the condition fails, we can employ loops.

`in` is a keyword in kotlin that works with the range of values or a collection.

The `for` loop

`..` -> defines a range
`1..10` -> It means 1 to 10

```
package com.nareshtech.scorekeeper

fun main()
{
    for (i in 1..10){
        print("$i ")
```

```
    }
}
```

```
package com.nareshtech.scorekeeper

fun main()
{
    for (i in 10 downTo 1){
        print("$i ")
    }
}
```

```
fun main()
{
    for (i in 1..10 step 2){
        print("$i ")
    }
}
```

```
package com.nareshtech.scorekeeper

fun main()
{
    for (i in 10 downTo 1 step 3){
        print("$i ")
    }
}
```

```
package com.nareshtech.scorekeeper

fun main()
{
   val strings = listOf("Pavan", "Kumar", "Reddy", "Tadi")
    for(i in strings){
        print("$i ")
    }
}
```

```
package com.nareshtech.scorekeeper

// Given a list of marks of the students obtained in an exam, calculate the
average marks of the list
fun main()
{
    val marks = listOf(85, 92, 78, 95, 88, 76, 90, 83, 98, 80, 65, 72, 89, 91,
79, 87, 93, 75, 82, 96,
        70, 84, 97, 81, 86, 73, 99, 77, 94, 68, 74, 86, 92, 71, 83, 88, 90, 78,
85, 69)

    // You want to find the average marks obtained by the class
    var sum = 0
    for (i in marks){
        sum += i
    }
    println("The average marks obtained by the class is ${sum/marks.size}")
    // You can find the highest mark scored in the entire class
    var max = marks[0]
    for(i in marks){
        if(i>max){
            max = i
        }
    }
    println("The highest score is $max")
    // you find the lowest mark scored in the entire class
}
```

while & do-while

```
package com.nareshtech.scorekeeper

// Given a list of marks of the students obtained in an exam, calculate the
average marks of the list
fun main()
{
```

```
    // you want to print 1 to 10 numbers
    var i = 1
    while (i<=10)
    {
        print("$i ")
        i++
    }

    println()

    do{
        print("$i ")
        i--
    }while(i>=1)
}
```

Assignment

- Write a program to find if a given number is prime or not
- Extend the above program to identify all the palindromic prime numbers till 10,000
- Write a program to idenfity if a given number fits in fibonacci series or not.
    - 0,1,1,2,3,5,8,13,21,34,55,89...
- Write a program to identify the factorial of any given number

***repeat* function in kotlin**

```
fun main()
{
   repeat(10){
       print("NIIT ")
   }
}
```