Smt. Chandaben Mohanbhai Patel Institute of Computer Applications

CA621
Basics of Mobile Applications
BSc.IT-Semester-VI

Unit-1
Introduction Android with Kotlin

Dr. Arpankumar Raval, Assistant Professor, CMPICA, CHARUSAT

Topics

1. Introduction to Mobile Applications
2. About Android
3. Basics of Kotlin
4. Setting up development environment
5. Dalvik Virtual Machine, Current is (ART) Android Run time
6. Android Application Structure- DDMS- Deprecated (Android profiler), Android Manifest File, APK
7. Gradle, Android permissions, Basic Building blocks – Activities, Services, Broadcast Receivers & Content providers

1. Introduction to Mobile Applications

- What is a Mobile Application?
  - Definition: A software application designed to run on mobile devices such as smartphones and tablets.
  - Examples: Social media apps (Instagram, Twitter, WhatsApp), Utility apps (Google Maps), and Entertainment apps (Spotify).
  - Importance: Enhances productivity, communication, and entertainment.

Mobile Application Development Options

- Mobile Application Development Options

1. Native Apps: Built for specific platforms (e.g., Android, iOS).

- Languages: Kotlin, Swift, Java.
- Pros: High performance and platform-specific features.
- Cons: Requires separate development for each platform.

2. Cross-Platform Apps: Single codebase for multiple platforms.

- Frameworks: Flutter, React Native, Xamarin.
- Pros: Cost-effective and faster development.
- Cons: Limited platform-specific features.

3. Web Apps: Runs in browsers; built using HTML, CSS, JavaScript.

- Pros: No installation needed; accessible on all devices.
- Cons: Limited offline functionality.

4. Hybrid Apps: Combines web and native app features.

- Frameworks: Ionic, PhoneGap.
- Pros: Cross-platform with added native features.
- Cons: Lower performance compared to native apps.

Tools for Mobile Application Development

1. Android Development
   - ❖ Android Studio, IntelliJ IDEA
2. iOS Development
   - ❖ Xcode, AppCode
3. Cross-Platform Development
   - ❖ Flutter (Android Studio), React Native (VS Code), Xamarin
4. **Web Development Tools**
   - ❖ Visual Studio Code, Sublime Text

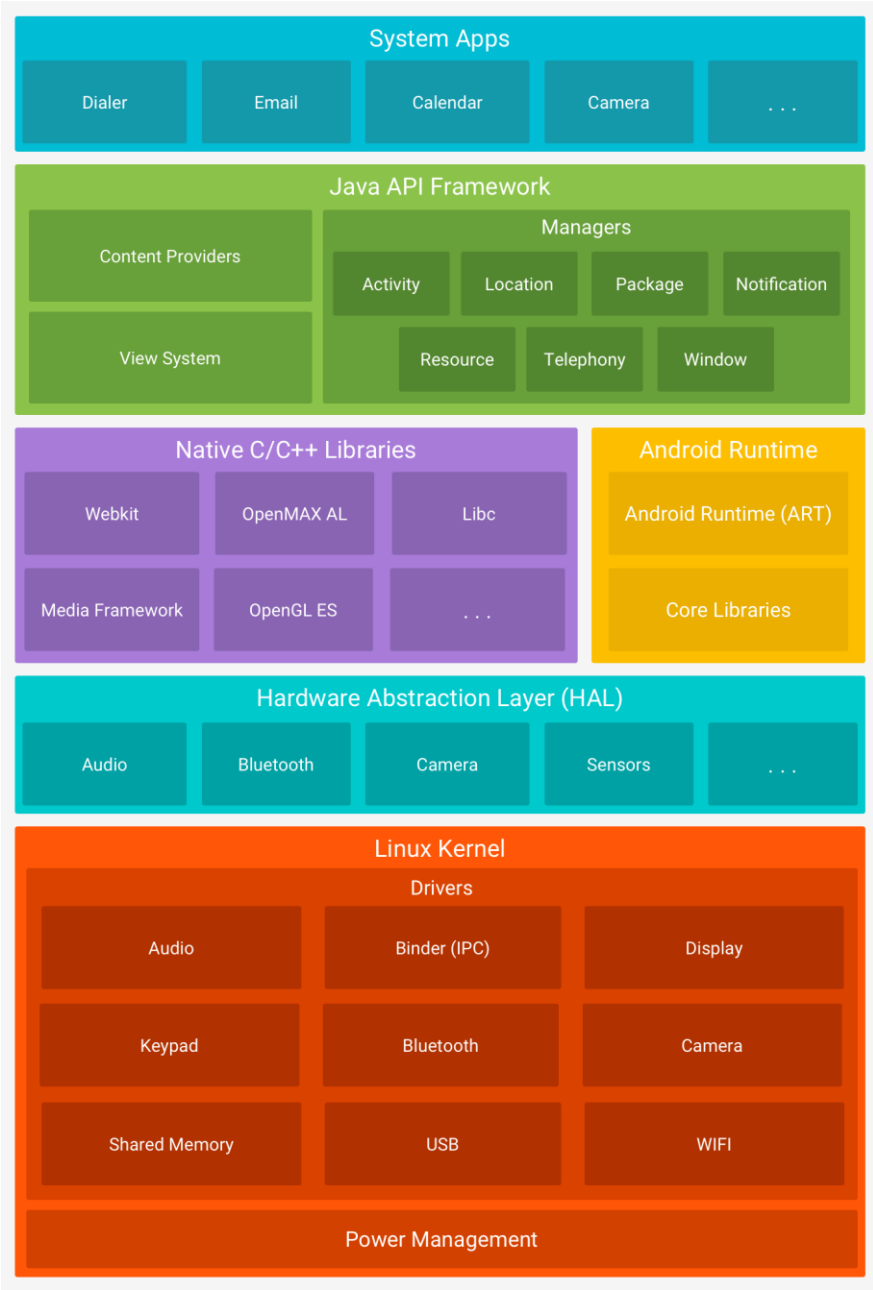
2. About Android

- Android is an open-source mobile operating system developed by Google.
- First release: **September 23, 2008** (Android 1.0).
- Key features: Open-source, customizable UI, and large developer community.
- Market share: Dominates the global smartphone market with over 70% share.

Android Development Software

- **Android SDK**: Introduced in 2007 alongside Android beta.
- **Android Studio**: Released in **May 2013**, replacing Eclipse.
- Features: Code editor, emulator.
- Testing tools, Gradle-based build system.
- **Java**: Initial Android Application Development Language.
- **Kotlin**: Official language for Android development since **May 2017**.

Android Architecture & Components



Key Components of Android

- **Linux Kernel:** Provides core system services like memory management, security, and hardware abstraction.
- **Android Runtime (ART):** Executes apps, ensuring efficient memory usage and performance.
- **Application Framework:** Offers APIs for building apps, like managing resources, views, and notifications.
- **Applications Layer:** Contains pre-installed apps like Contacts, Messages, and custom user applications.

Features of the OS

1. **Multitasking:** Run multiple apps simultaneously.
2. **Customizability:** Themes and widgets, Diverse launchers for a personalized user experience.
3. **Wide Hardware Support:** Works across low-end to high-end devices.
4. **Google Ecosystem:** Integration with services like Google Maps, Google Drive, and Play Store.
5. **Rich Developer Community:** Constant updates and enhancements due to open-source contributions.

Current Trends in Android

- **5G Integration**: Enhancing connectivity and app performance.
- **AI and Machine Learning**: Smart suggestions, voice assistants, and real-time translation.
- **Android for IoT**: Expanding use in smart TVs, wearables, and connected devices.
- **Material Design 3**: Modern UI design guidelines with adaptive themes.
- **Security Features**: Improved app permissions, biometric authentication, and regular updates.

Android OS Versions


Apple Pie 1.0


Cupcake 1.5


Donut 1.6


Eclair 2.0/ 2.1


Froyo 2.2


Gingerbread 2.3.x


Honeycomb 3.x


Ice Cream Sandwich 4.0.x


Jelly Bean 4.1/4.2/4.3


KitKat 4.4


Lollipop 5.0


Marshmallow 6.0


Nougat 7.0


Oreo 8.0


Pie 9.0

- Android Apple 1.0 in April 2009 to Pie 9.0 in August 2018

Latest Versions

- 10 Q — API Level 29
- 11 Red Velvet Cake — API Level 30
- 12 Snow Cone — API level 31,32
- 13 Tiramisu — (15th August 2022) API level 33
- 14 Upside down cake — (4th October 2023) API level 34
- 15 Vanilla Ice cream — (15th October 2024) API level 35
- 16 Baklava — (19th November 2024) API level 36)

3. Basics of Kotlin

- **Kotlin** is a statically-typed programming language, developed by JetBrains.
- **An entry point of a Kotlin application is the main function**.

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

- **Where to run the code:** In Android Studio> New Project > On com.example.proj> Right Click> New> Kotlin Class/File > give class name > remove Class > write fun main() {print(“Hi”)} and run

Kotlin Print function and Variables

Print function in Kotlin

- To output something on the screen the following two methods are used:
- **print() :** The print statement prints everything inside it onto the screen.
- **println() :** The println statement appends a newline at the end of the output.

Variables:

- There are two types of variables – **mutable and immutable.**
- An immutable variable is one whose value cannot be changed, also known as unchangeable or read-only variable, on the other hand the value of the mutable variable can be changed.

Mutable Immutable variables

- Immutable variable is declared using **val** keyword in Kotlin. It will assigns the type based on the type of value provided

  val myName = “ABC“

- Mutable Variable allows to change the value. In Kotlin, we use **var** keyword to declare a mutable variable.

  var myName = “Arpan"

Type inference

- We can declare and initialize the variable in a single statement like this:

  var website = "beginnersbook"

- In the above statement we have not specified the type of the variable, kotlin knows that the variable website is a string. Compiler can understand the type of the variable by looking at the values.
- However if you want to explicitly mention the type of the variable then you can do that like this:

  var website: String = "beginnersbook“

  val num: Int = 123456

- Here we have explicitly mentioned the type of variable “website” as String.

Get String Length in Kotlin

var name = “Arpan” val len=name.length, print(len) o/p: 5

- Compare Strings in Kotlin

  var str1 = "BeginnersBook"
  var str2 = "beginnersbook"

  str1.equals(str2) will return false can be used in case of comparison of the strings we can use it with the if condition.

- Access character in a string at a specific index

  str.get(3) It will print the character no 4 as the index starts from 0

Safe Calls

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).

Safe call operator, written '?.'

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

- This returns b.length if b is not null, and null otherwise. The type of this expression is Int?

- A regular variable of type String can not hold null:
  - var a:String = "abc"
  - a = null // compilation error
- To allow null, we can add a? after the data type of that property which declares that variable as a nullable property
  - **var b: String? = "abc"**
  - **b = null // ok**
  - **print(b)**

Kotlin Data Types

1. Numbers – Byte, Short, Int, Long, Float, Double
2. Boolean – True, false
3. Characters
4. Arrays
5. Strings

Read values in Kotlin

- import java.util.Scanner

  var reader = Scanner(System.`in`)

  - reader.nextInt()
  - reader.nextFloat()
  - reader.nextLine()
  - reader.nextChar()
  - reader.nextDouble()

Kotlin – How to take Input from User

- **readLine**() function to read the string entered on console.

```
import java.util.Scanner
fun main()
{
  val reader =   Scanner(System.`in`)
  print("Write anything here: ")
  val enteredString = reader.readLine()
  println(enteredString)
   }
```

Kotlin : type conversion.

1. **toChar()** – To convert a type to Char type.
2. **toInt()** – To convert a type to Int type.
3. **toLong()** – To convert a type to Long type.
4. **toFloat()** – To convert a type to Float type.
5. **toDouble()** – To convert a type to Double type.
6. **toByte()** – To convert a type to Byte type.
7. **toShort()** – To convert a type to Short type.

Arrays

- Arrays in Kotlin are able to store multiple values of different data types.
- We can access element of array using its index.
- **Kotlin Array Declaration**
- Array that holds multiple different data types.

```
var arr = arrayOf(10, "BeginnersBook", 10.99, 'A’)
```

- Array that can only hold integers

```
var arr = arrayOf<Int>(1, 22, 55)
```

- Array that can only hold strings

```
var arr2 = arrayOf<String>("ab", "bc", "cd")
```

- Get size of array : arr.size
- Check the element in an array : arr.contains(“Arpan")

```
var arr = arrayOf(“Arpan”,10,10.30,90898989,’D’)
Println(arr.size)    o/p = 5
If(arr.contains(“Arpan”))
{ print(“yes”)}
else
{print(“no”)}
```

Kotlin If – Else Expression

```
if(condition){
   // Statements that need to be executed if condition is true
   ...
}
```

```
fun main() {

    val number = 100

if (number%2 == 0)
{
        println("Even Number")
}
```

Kotlin – If..Else expression

```
fun main(args: Array<String>) {
    // Marks out of 100
    val marks = 90
    if (marks < 30) {
        println("Father will get angry")
    }
    else {
        println("Father will be happy")
    }
}
```

```
var max: Int
if (a > b) {
    max = a
} else {
    max = b
}
```

If..else if..else expression

```
fun main(args: Array<String>) {
    val num = 99
    if(num<0)
        println("Number is Negative")
    else if (num>0 && num<10)
        println("Single digit number")
    else if (num>=10 && num <100)
        println("Double digit number")
    else
        println("Number has 3 or more digits")
}
```

Kotlin - When expression

When expression in Kotlin works same as switch case in other programming languages

```
fun main(args : Array<String>){
    var ch = 'A’
     when(ch){
        ‘A' -> println("A is a Vowel")
        'E' -> println("E is a Vowel")
        'I' -> println("I is a Vowel")
        'O' -> println("O is a Vowel")
        'U' -> println("U is a Vowel")
 else -> println("$ch is a Consonant")
    }
}
```

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

    var ch = 'A'

    when(ch){

        'A', 'E', 'I', 'O', 'U' -> println("$ch is a
Vowel")

        else -> println("$ch is a Consonant")
    }
}
```

When expression with ranges

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

    var num = 78
    when(num) {
        in 1..9 -> println("$num is a single digit number")
        in 10..99 -> println("$num is a two digit number")
        in 100..999 -> println("$num is a three digit number")
        else -> println("$num has more than three digits")
    }
}
```

Kotlin for Loop and for Array

- fun main(args : Array<String>){
  for(n in 10..15){
  println("Loop: $n")
  } }

```
fun main(args : Array<String>){
    val myArray = arrayOf("ab", "bc", "cd", "da")
    for (str in myArray){
        println(str)
    }
}
```

While loop

```
fun main(args : Array<String>){
    var num = 10
    while(num>=5){
        println("Loop: $num")
        num--
    }
}
```

Kotlin for loop iterating though array indices

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

    val myArray = arrayOf("Steve", "Robin", "Kate", "Lucy")
    for (n in myArray.indices){
        println("myArray[$n]: ${myArray[n]}")
    }
}
```

Kotlin Function

Syntax:

fun ***function_name***(*param1: data_type, param2: data_type, ...*): *return_type*

Example:

```
fun sayHello()
{
    println("Hello")
}
```

Function Examples

```
fun addNumbers(n1: Double, n2: Double): Int {
    val sum = n1 + n2
    val sumInteger = sum.toInt()
    return sumInteger
}
fun main(args: Array<String>) {
    val number1 = 12.2
    val number2 = 3.4
    val result: Int
    result = addNumbers(number1, number2)
    println("result = $result")
}
```

You can omit the curly braces { } of the function body and specify the body after = symbol if the function returns a single expression (like below example).

```
fun main(args: Array<String>) {
    println(getName("John", "Doe"))
}

fun getName(firstName: String, lastName: String): String = "$firstName $lastName"
```

Try, catch and throw

```
try {
    //code where an exception can occur
}
catch (e: SomeException) {
    // handle the exception
}
finally {
    // optional block but executes always
}
```

Try, catch and throw

```
fun main(args: Array<String>) {
    try{
        var num = 100/0
        println(num)
    }
    catch(e: ArithmeticException){
        println("Arithmetic Error in the code")

    }
    println("Out of try catch block")
}
```

Try Catch with Character Input

```
import java.util.Scanner
fun main(args: Array<String>) {
    val scanner = Scanner(System.`in`)
    print("Enter A,B or C for Input:")
    try{
    val chr = scanner.next().single()
    print(chr)
    }
    catch(e:Exception)
    {
        print("Only a character like A, B or C")
    }
}
```

Kotlin with OOP

Class in Kotlin

In Object-Oriented Programming (OOP), a class is a blueprint or template for creating objects. It defines a set of attributes (data members) and methods (functions) that represent the state and behavior of the objects created from it. By default all the classes are Final in the Kotlin (To avoid the fragile base class Problem)

```
public class MyClass {
  // variables or data members
  // member functions
  .. ..
}
```

```
class Example {    // data member
    private var number: Int = 5
    // member function
    fun calculateSquare(): Int {
        return number*number
    } }
```

Create object of the class and access the Members and Member functions

Create the Object named e for the class Example

- var e = Example()

//Access data member

e.number

//Access member function

e.calculateSquare()

Program of Example as a class

```
class Example {
    // data member
    private var number: Int = 5
    // member function
    fun calculateSquare(): Int {
        return number*number
    }
}
fun main(args: Array<String>) {
    // create obj object of Example class
    val obj = Example()
    println("${obj.calculateSquare()}")
}
```

Kotlin Constructors

- In Kotlin, we have two types of constructors – primary and secondary constructor.

1. Primary Constructor – Initialize the properties of the class in the header (Automatically called), Can be only one at a time.

2. Secondary Constructor – Initialize the properties of class, we can have additional initialization code inside secondary constructor. We may have more than one constructors. By using the keyword constructor.

Primary Constructor

The primary constructor is part of the class header.

```
class Student(val name: String, var age: Int) {
    //This is my class. For now I am leaving it empty
}
fun main(args: Array<String>) {
    //creating the object of class Student
    val stu = Student(“Arpan", 31)
    println("Student Name: ${stu.name}")
    println("Student Age: ${stu.age}")
}
```

Default values in Kotlin constructor

```
class Student(val name: String = "Student", var age: Int = 99) {
    //This is my class. For now I am leaving it empty
}
fun main(args: Array<String>) {
    //creating the object of class Student
    val stu = Student(“ABC", 31)
    val stu2 = Student(" ABC ")
    val stu3 = Student()
    println("Name: ${stu.name} and Age: ${stu.age}")
    println("Name: ${stu2.name} and Age: ${stu2.age}")
    println("Name: ${stu3.name} and Age: ${stu3.age}")
}
```

Initializer Block inside Kotlin Constructor

```
class Student(val name: String = “Arpan", var age: Int = 30)
{    val stuName: String
     var stuAge: Int
   init{
      if(name == “Arpan") {
         stuName = "NA"
         stuAge = 0
   }
      else {
         stuName = name.toUpperCase()
         stuAge = age
      }  println("Student Name is: $stuName")
         println("Student Age is: $stuAge")
   }   }
```

- **The primary constructor has a constrained syntax, and cannot contain any code.**
- To put the **initialization** code, initializer block is used. It is prefixed with **init keyword.**

```
fun main(args: Array<String>) {
    val person1 = Person("joe", 25)
}
class Person(fName: String, personAge: Int)
 { val firstName: String
    var age: Int
    // initializer block
    init {
        firstName = fName.capitalize()
        age = personAge
        println("First Name = $firstName")
        println("Age = $age")
    }
}
```

Kotlin Secondary Constructor

- A class can also contain one or more secondary constructors. They are created using **constructor** keyword.
- Secondary constructor comes up when you need to extend a class that provides multiple constructors that initialize the class in different ways.
- Can be only identified with different arguments.

Example

```
class Log {
    constructor(data: String) {
        // some code
    }
    constructor(data: String, numberOfData: Int) {
        // some code
    }
}
```

Example

```
fun main(args: Array<String>){
    val obj = Student (“Arpan", 30)
}
class Student{
    constructor(name: String, age: Int){
        println("Student Name: ${name.toUpperCase()}")
        println("Student Age: $age")
    }
}
```

Kotlin Inheritance

- By default **all classes in Kotlin are final** so you have to **use the open annotation in the parent class**, this tells the compiler that this class can be inherited by other classes. Otherwise it will not allows the main class to be inherited. It is final to avoid **fragile** base class error.
- Advantage of inheritance: Reusability of the code and to add more features by derived class as well as to use the features from the base class.

```
open class ParentClass {
        val a=5
        fun show(){
                println("This is base class function")
}
}

class ChildClass( ): ParentClass( ){
        fun calculate(){
        println(a * a)   }
        }
}
fun main(){ var cc = ChildClass() cc.calculate() cc.show()}
```

Override the member

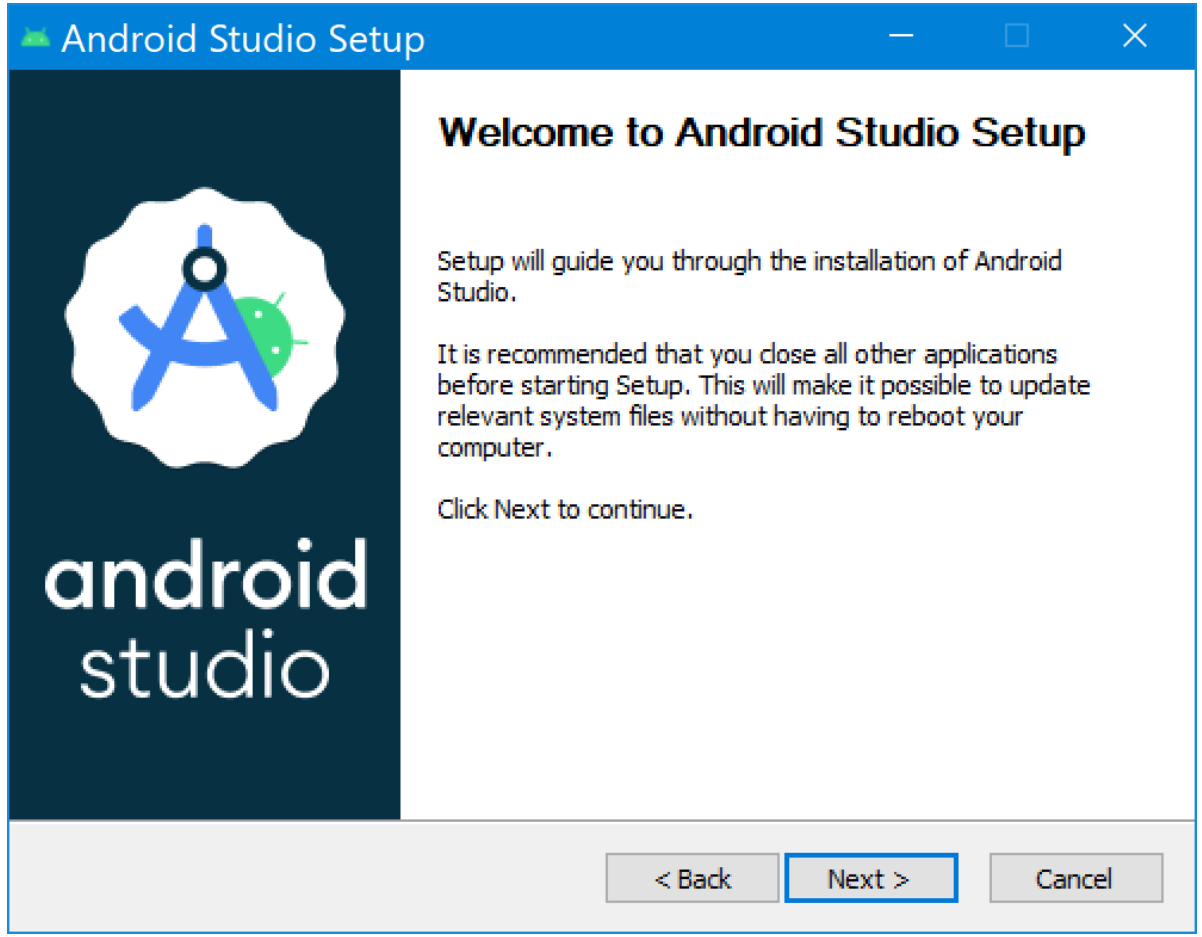
- A **child class can override the member function of parent class** using **override** keyword.
- By overriding a function, a child class gives its own implementation to the already existing code of base class.

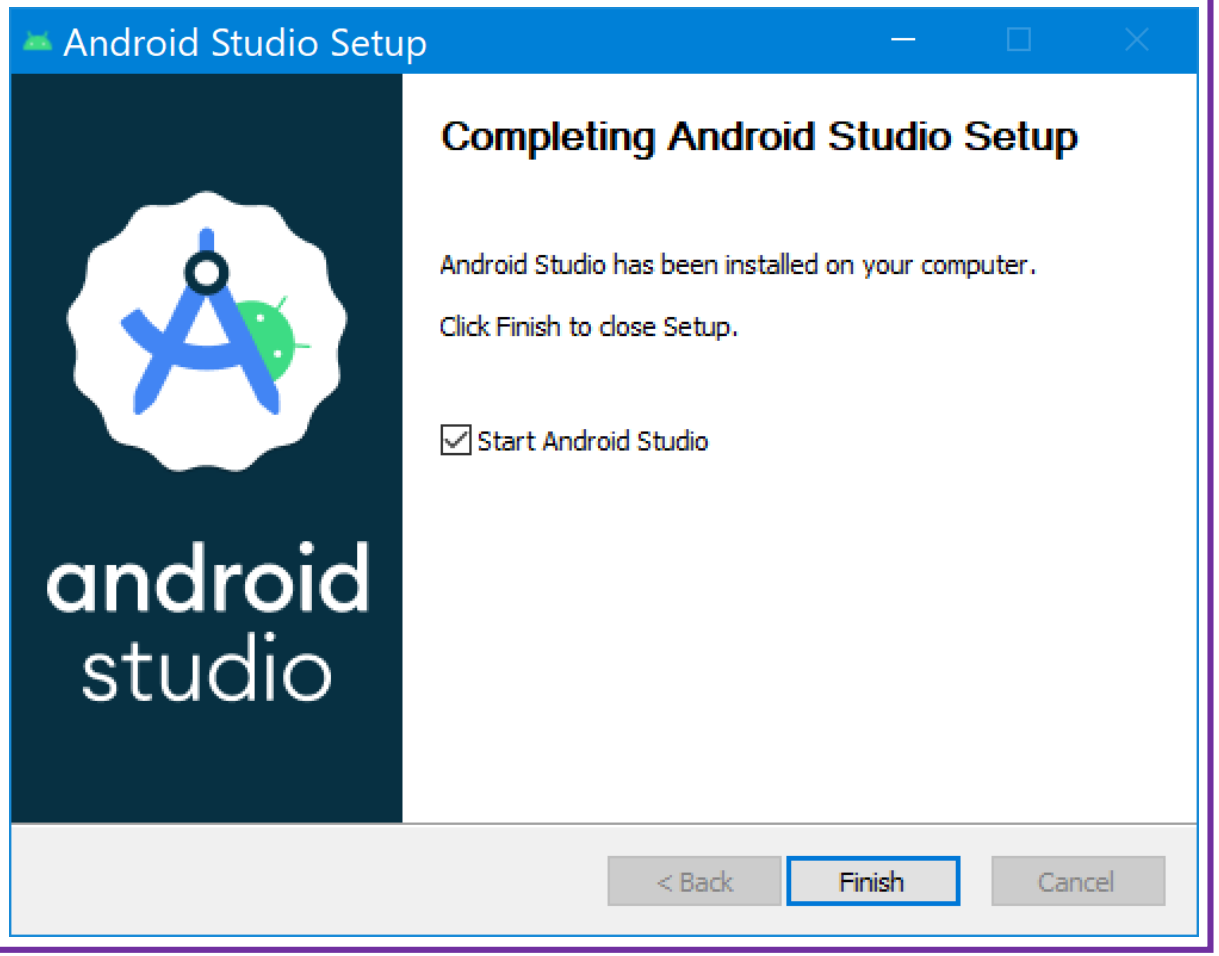
Example

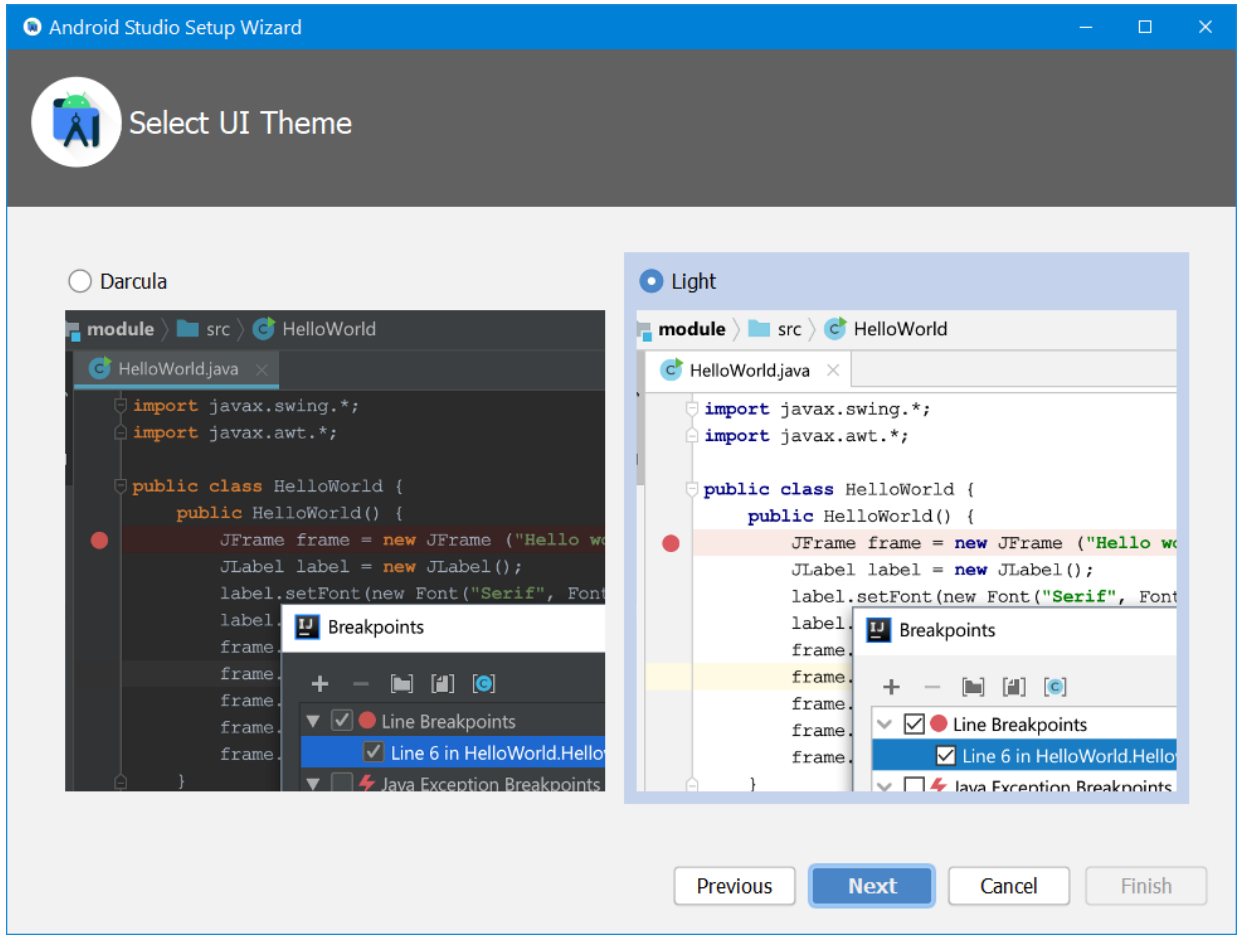
```
open class Animal {
    open fun sound() {
        println("Animal makes a sound")
    } }
class Dog: Animal() {
    override fun sound() {
        println("Dog makes a sound of woof woof")
    }
}
fun main(args: Array<String>) {
    val d = Dog()
    d.sound()
}
```

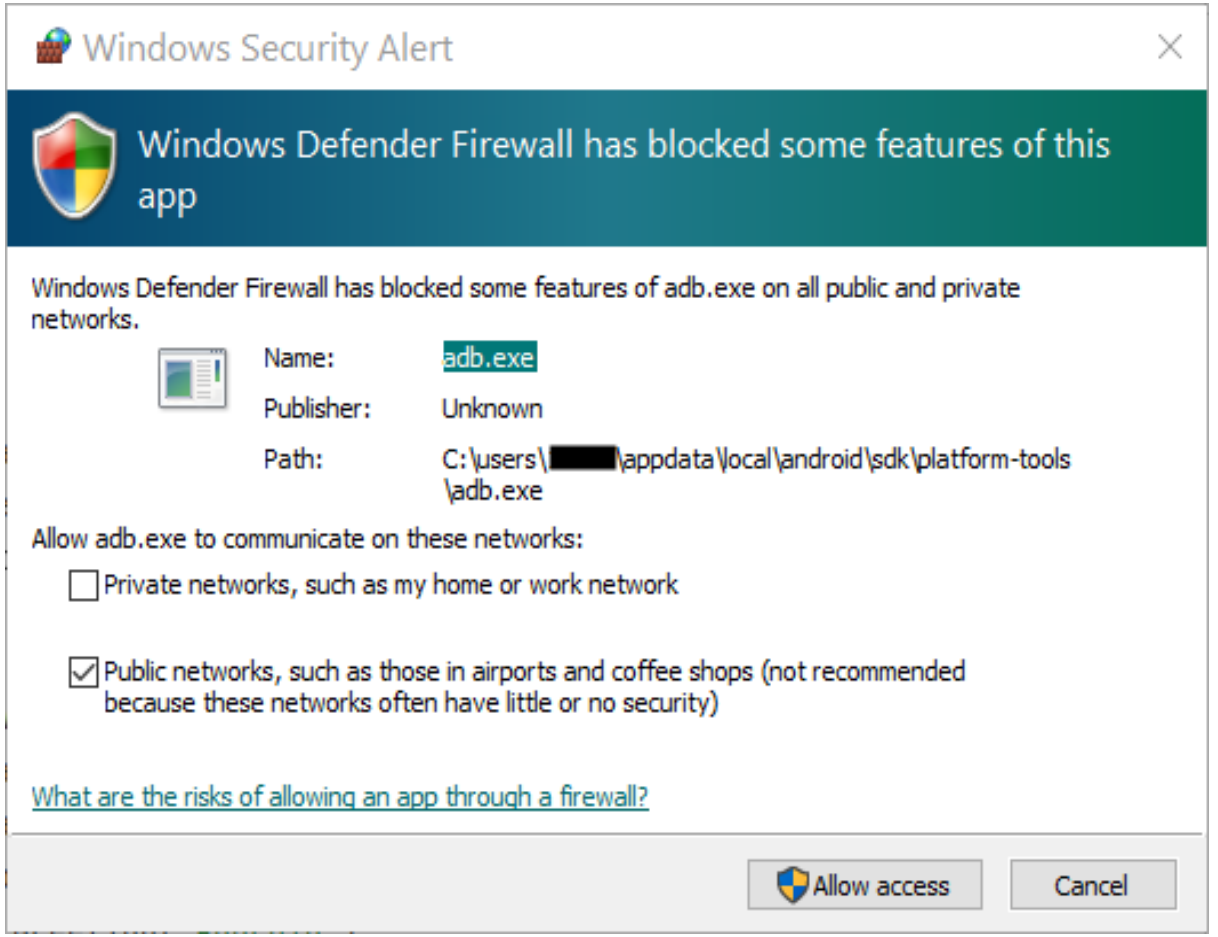
4. Setting up the Development Environment

- Requirements:
  8 GB and Above RAM, 1280 x 800 Minimum Screen Resolution, 512 GB of ROM, 64-Bit 8/10/11 Windows OS
- Download Android Studio from https://developer.android.com
- Choose for Windows and start download after the Agreement to the license
- Follow the steps now
- For new Project After Installment: + sign New Project > Choose Empty Views Activity (in Older version) Empty Activity > Provide Name only and in latest version keep API 30 Min and Finish


Android Studio Setup
Welcome to Android Studio Setup
Setup will guide you through the installation of Android Studio.
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.
Click Next to continue.
android studio
< Back
Next >
Cancel

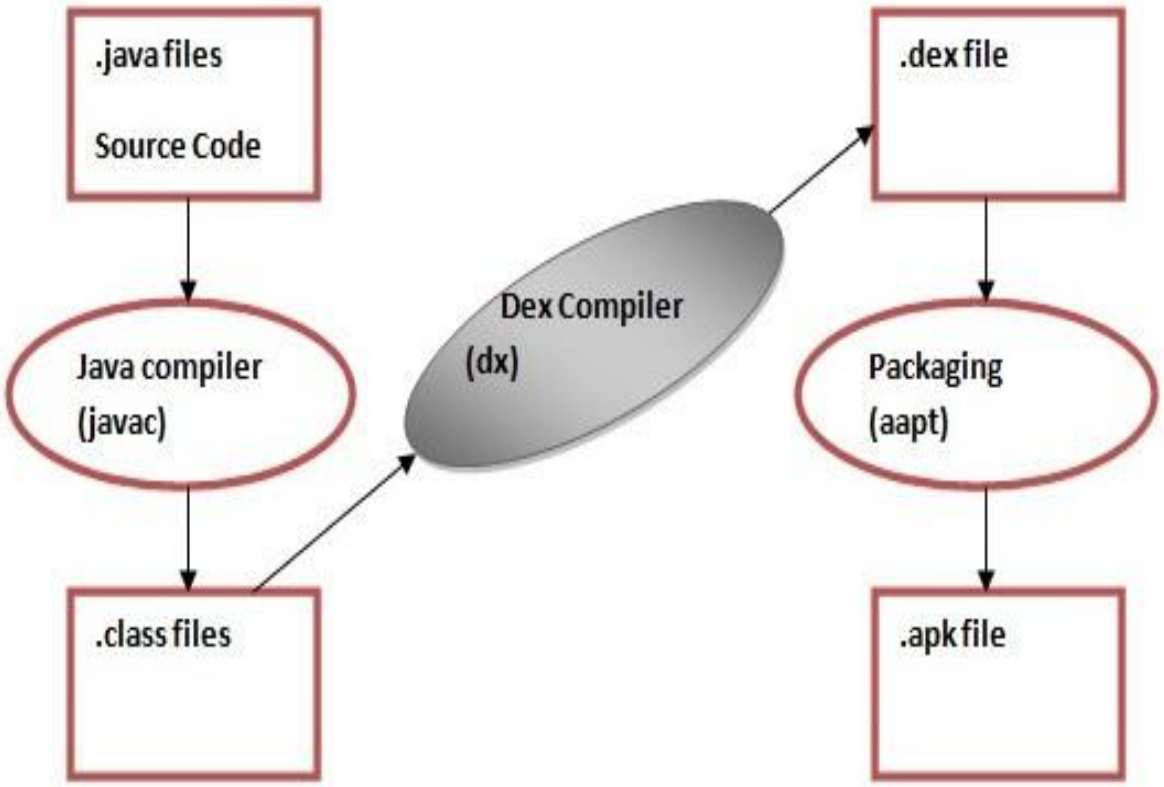

Android Studio Setup
Completing Android Studio Setup
Android Studio has been installed on your computer.
Click Finish to close Setup.
Start Android Studio
android studio
< Back
Finish
Cancel


Android Studio Setup Wizard
Select UI Theme
Darcula
Light
module src HelloWorld
HelloWorld.java
import javax.swing.*;
import javax.awt.*;
public class HelloWorld {
public HelloWorld() {
JFrame frame = new JFrame ("Hello w
JLabel label = new JLabel();
label.setFont(new Font("Serif", Font
Breakpoints
Line Breakpoints
Line 6 in HelloWorld.Hello
Java Exception Breakpoints
Previous
Next
Cancel
Finish


Windows Security Alert
Windows Defender Firewall has blocked some features of this app
Windows Defender Firewall has blocked some features of adb.exe on all public and private networks.
Name: adb.exe
Publisher: Unknown
Path: C:\users\[redacted]\appdata\local\android\sdk\platform-tools\adb.exe
Allow adb.exe to communicate on these networks:
Private networks, such as my home or work network
Public networks, such as those in airports and coffee shops (not recommended because these networks often have little or no security)
What are the risks of allowing an app through a firewall?
Allow access
Cancel

5. Dalvik Virtual Machine, Current is (ART) Android Run time

- Dalvik Virtual Machine: The Dalvik Virtual Machine (DVM) is an android virtual machine optimized for mobile devices. It optimizes the virtual machine for memory, battery life and performance.
- The Dex compiler converts the class files into the .dex file that run on the Dalvik VM. Multiple class files are converted into one dex file.
- This format allows the apps to run quickly with fewer resources, i.e. on mobile phones and low-memory, slower devices. This is different from a typical Java Virtual Machine (JVM), as it optimized especially for Android to run apps that use less memory.

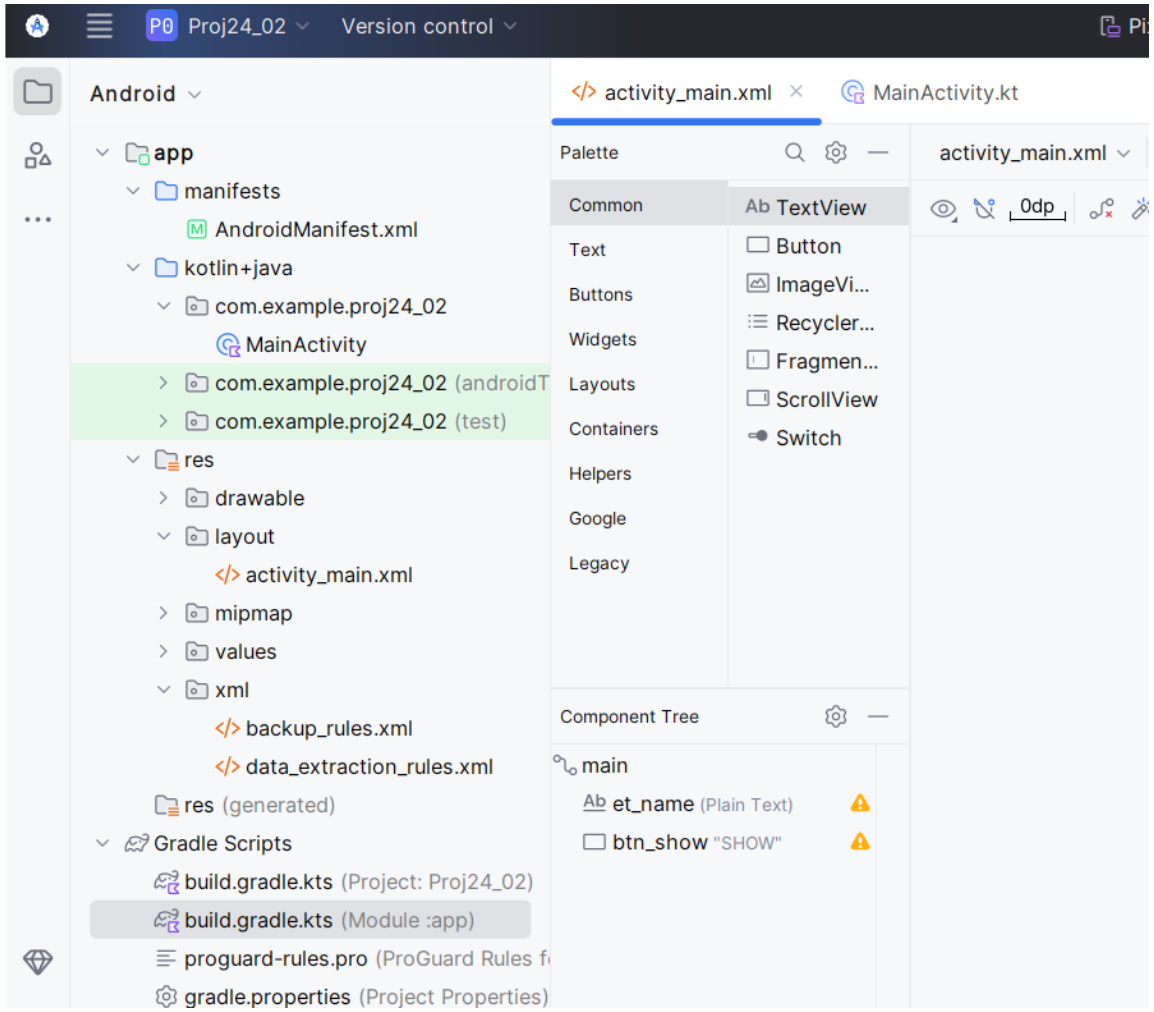

.java files
Source Code
Java compiler
(javac)
.class files
Dex Compiler
(dx)
.dex file
Packaging
(aapt)
.apk file

ART

- Android Runtime (ART) is an application runtime environment used by the Android operating system.
- Replacing Dalvik, the process virtual machine originally used by Android, ART performs the translation of the application's bytecode into native instructions that are later executed by the device's runtime environment.
- Features:

1. Ahead of Time, ART introduces ahead-of-time (AOT) compilation, which can improve app performance. ART also has tighter install-time verification than Dalvik.
2. Improved Garbage Collection: Regarding Memory allocation and time.
3. Development and debugging improvements.

6. Android Application Structure



1. **AndroidManifest.xml**: Every project in Android includes a manifest file, which is AndroidManifest.xml, stored in the root directory of its project hierarchy. This file includes nodes for each of the Activities, Services, Content Providers and Broadcast Receiver that make the application and using Intent Filters and Permissions, determines how they co-ordinate with each other and other applications.
2. **Kotlin + Java**: The Java folder contains the Java source code files. These files are used as a controller for controlled UI (Layout file). It gets the data from the Layout file and after processing that data output will be shown in the UI layout. It works on the backend of an Android application.
3. **Drawable**: A Drawable folder contains resource type file (something that can be drawn). Drawables may take a variety of file like Bitmap (PNG, JPEG), Nine Patch, Vector (XML), Shape, Layers, States, Levels, and Scale.

4. **Layout**: A layout defines the visual structure for a user interface, such as the UI for an Android application. This folder stores Layout files that are written in XML language. You can add additional layout objects or widgets as child elements to gradually build a View hierarchy that defines your layout file.
5. **Mipmap**: Mipmap folder contains the Image Asset file that can be used in Android Studio application. You can generate the following icon types like Launcher icons, Action bar and tab icons, and Notification icons.
6. **Colors.xml**: colors.xml file contains color resources of the Android application. Different color values are identified by a unique name that can be used in the Android application program.Below is a sample colors.xml file:

7. **Strings.xml**: The strings.xml file contains string resources of the Android application. The different string value is identified by a unique name that can be used in the Android application program. This file also stores string array by using XML language.
8. **Styles.xml**: The styles.xml file contains resources of the theme style in the Android application. This file is written in XML language.
9. **Build.gradle(Module: app)**: This defines the module-specific build configurations. Here you can add dependencies what you need in your Android application.
10. **build.gradle(Module: app)**: This defines the module-specific build configurations. Here you can add dependencies what you need in your Android application.

DDMS / Android Profiler

- Dalvik Debug Monitor Server (DDMS) is a debugging tool in Android that helps developers develop and test Android applications. It's included in the Android SDK and is considered one of the most important introspection tools for Android developers.
- This tool is deprecated. Instead, use Android Profiler in Android Studio 3.0 and higher to profile your app's CPU, memory, and network usage.
- Fixing performance problems involves *profiling* your app, or identifying areas in which your app makes inefficient use of resources such as the CPU, memory, graphics, or the device battery.

Manifest File and APK

- The **manifest** file is required to declare the following:
- The components of the app, including all activities, services, broadcast receivers, and content providers. Each component must define basic properties, such as the name of its Kotlin or Java class. It can also declare capabilities, such as which device configurations it can handle, and intent filters that describe how the component can be started.
- An Android package, which is an archive file with an . **Apk** suffix (Extension) , contains the contents of an Android app required at runtime, and it is the file that Android-powered devices use to install the app.

7. Gradle, Android Permissions, Basic Building Blocks

- **Gradle:** Gradle is a build system that helps to automate and manage your building process. It downloads required dependencies, packages your code, and prepares it for compilation.

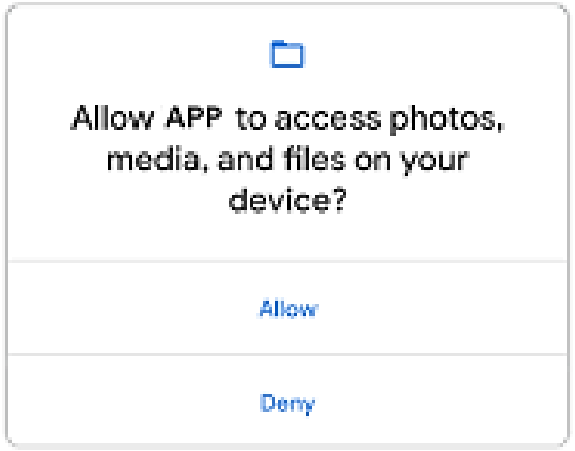
  Gradle Kotlin DSL. The Gradle Kotlin DSL is a domain specific language that you can use to write build scripts quickly and efficiently.

- **Android Permissions:** It provide controls that increase user awareness and limit an app's access to sensitive data.

1. **Install Time-Permissions**, Normal Permission: allow access to data and actions, Signature: signed by the same certificate.

2. Runtime permissions: also known as dangerous permissions, give your app additional access to restricted data.

3. Special Permissions: when they want to protect access to particularly powerful actions, such as drawing over other apps



Basic Building Blocks

- **Activities**: An activity is a fundamental component of an Android app that represents a single screen where users can interact. One activity contains two files Code .KT and Design .XML
- **Services:** In Android, a service is a component that can run in the background to perform tasks without a user interface: Handling network transactions, Playing music, Performing file I/O, and Interacting with a content provider.
- **Broadcast Receivers & Content Providers**: Broadcast in android is the system-wide events that can occur when the device starts, when a message is received on the device or when incoming calls are received, or when a device goes to airplane mode, etc.

- Broadcast Receivers are used to respond to these system-wide events. Broadcast Receivers allow us to register for the system and application events, and when that event happens, then the register receivers get notified. There are mainly two types of Broadcast Receivers:
- **Static Broadcast Receivers:** These types of Receivers are declared in the manifest file and works even if the app is closed.
- **Dynamic Broadcast Receivers:** These types of receivers work only if the app is active or minimized.

Intent	Description Of Event
android.intent.action.BATTERY_LOW :	Indicates low battery condition on the device.
android.intent.action.BOOT_COMPLETED	This is broadcast once after the system has finished booting
android.intent.action.CALL	To perform a call to someone specified by the data
android.intent.action.DATE_CHANGED	Indicates that the date has changed
android.intent.action.REBOOT	Indicates that the device has been a reboot
android.net.conn.CONNECTIVITY_CHANGE	The mobile network or wifi connection is changed(or reset)
android.intent.ACTION_AIRPLANE_MODE_CHANGED	This indicates that airplane mode has been switched on or off.

Content Providers

- A content provider manages access to a central repository of data. A provider is part of an Android application, which often provides its own UI for working with the data. However, content providers are primarily used by other applications, which access the provider using a provider client object.
- How content providers work.
- The API you use to retrieve data from a content provider.
- The API you use to insert, update, or delete data in a content provider.
- Other API features that facilitate working with providers.


THANK
YOU