Introduction to Android Development

Department of Computer Engineering - Al-Azhar University - Egypt

Objective

The objective of this lab is to introduce students to the fundamentals of android development using Koltlin and XML
Students will learn how to create a simple android app and will learn basic concepts of android development

Introduction :

Welcome to the exciting world of Android Development! In our rapidly evolving digital era, mobile applications play a pivotal role in shaping the way we interact with technology. The Android Development Lab is designed to provide you with a comprehensive and hands-on experience in creating mobile applications for the Android platform.

Why Android Development?

Android, being the most widely used mobile operating system globally, offers an incredible opportunity for developers to bring their innovative ideas to life. Whether you aspire to create user-friendly applications, explore the realms of game development, or delve into emerging technologies such as augmented reality, Android provides a versatile and dynamic platform.

What to Expect?

Throughout this lab, you will embark on a journey that covers the fundamental concepts of Android development, from the basics of the Android Studio IDE to building interactive and functional applications. Our carefully crafted curriculum will guide you through essential topics such as user interface design, data storage, networking, and integration of key Android features.

Key Highlights:

Hands-on Experience: Learn by doing! Our lab is structured to ensure that you gain practical experience through a series of engaging exercises and projects.
Industry-Relevant Skills: Acquire skills that are highly sought after in the tech industry. Android development is not just about coding; it's about problem-solving, creativity, and building user-centric solutions.

Collaborative Learning: Engage with your peers, share ideas, and collaborate on projects. The lab is not just about individual growth but also fostering teamwork and communication skills.

Guest Lectures and Industry Insights: Stay updated with the latest trends and industry practices through guest lectures and discussions with professionals from the field of Android development.

Prerequisites:

No prior experience in Android development is required, but a basic understanding of programming concepts will be beneficial. Come with enthusiasm, an open mind, and a willingness to explore the possibilities of mobile app development.

By the end of this lab, you will have the skills and confidence to develop your Android applications, turning your creative ideas into tangible, functioning projects. Get ready to embark on a rewarding journey into the world of Android development!

Lets begin the adventure

What is Android Development?

Android development refers to the process of creating applications that run on the Android operating system, which is widely used in mobile devices such as smartphones and tablets. Developed by the Open Handset Alliance and later acquired by Google, Android has become the dominant mobile platform, providing a versatile ecosystem for developers to build innovative and user-friendly applications.

Key Components of Android Development:

Android Studio:
The official Integrated Development Environment (IDE) for Android development, offering a feature-rich environment for coding, testing, and debugging applications.\

Java/Kotlin Programming:
Android apps are primarily developed using Java or Kotlin programming languages, allowing developers to create robust and scalable applications.

User Interface (UI) Design:
Android provides a flexible UI framework for creating visually appealing and responsive user interfaces. XML is commonly used for layout design.

Activities and Fragments:
Android applications are built around activities and fragments, representing different screens or components of the app.

Intents:
Intents facilitate communication between different components of an Android app and allow the launch of activities or services.

Android SDK (Software Development Kit):
A comprehensive set of tools, libraries, and APIs provided by Google for developers to create Android applications.

Data Storage:
Android supports various methods of data storage, including SQLite databases, SharedPreferences, and file storage, enabling efficient data management.

Networking:
Integration of network-related tasks, such as making HTTP requests and handling responses, is crucial for apps that require data from the internet.

Security and Permissions:
Android incorporates security features to protect user data and privacy, and developers must adhere to a permission system to access certain device functionalities.

Testing and Debugging:
Robust testing and debugging tools within Android Studio help developers identify and rectify issues, ensuring the reliability of their applications.

Why Android Development Matters ?

Global Market Reach:
Android's widespread adoption means that apps developed for this platform have the potential to reach a vast and diverse global audience.

Open Source Ecosystem:
Android's open-source nature encourages innovation and collaboration, allowing developers to customize and enhance the platform.

Diverse App Categories:
Android accommodates a wide range of app categories, including games, productivity tools, social networking, and utilities, fostering creativity and diversity in app development.

Monetization Opportunities:
Developers can monetize their Android applications through various channels, such as in-app purchases, advertisements, and subscription models.

Continuous Evolution:
Android continually evolves with updates and new features, providing developers with opportunities to leverage the latest technologies and capabilities.

Tools and Resources:

Developers often use Android Studio, version control systems like Git, and Android Developer documentation to create high-quality apps. Additionally, engagement with the Android developer community and participation in forums contribute to skill enhancement and problem-solving.

In summary, Android development is an engaging and dynamic process that empowers developers to craft versatile and impactful applications for a global audience using the Android operating system.

Types of Android Development Methods

Android development encompasses various methods tailored to meet diverse needs and preferences. Developers can choose the approach that best aligns with their project requirements, team skills, and development goals. Here are the primary types of Android development methods:

1. Native Android Development:

Definition:

Native development involves creating applications specifically for the Android platform using official programming languages and tools.

Key Components:

Java and Kotlin programming languages.
Android Studio IDE.
Access to the full range of Android APIs.

Advantages:

Optimal performance and responsiveness.
Full access to device features.
Seamless integration with the Android ecosystem.

Considerations:

Requires separate development for other platforms.
Potentially longer development timelines.

2. Cross-Platform Development:

Definition:

Cross-platform development allows developers to create applications that run on multiple platforms, including Android, using a single codebase.

Key Components:

Frameworks such as React Native, Flutter, Xamarin.
Shared codebase for Android and other platforms.

Advantages:

Code reusability across different platforms.
Faster development cycles.
Consistent user experience.

Considerations:

May encounter limitations in accessing certain native features.
Slight performance trade-offs compared to native development.

3. Progressive Web Apps (PWAs):

Definition:

PWAs are web applications designed to function like native apps, accessible through web browsers on Android devices.

Key Components:

Web technologies (HTML, CSS, JavaScript).
Service workers for offline functionality.
Manifest files for installation.

Advantages:

Cross-platform compatibility.
No need for app store distribution.
Responsive design for various screen sizes.

Considerations:

Limited access to native device features.
May not provide the same level of performance as native apps.

4. Hybrid App Development:

Definition:

Hybrid apps combine elements of both native and web applications, typically using a web view to render web content within a native app shell.

Key Components:

HTML, CSS, JavaScript.
Frameworks like Apache Cordova, Ionic.

Advantages:

Single codebase for multiple platforms.
Faster development compared to native.
Access to certain native features.

Considerations:

May face performance challenges.
Dependency on web views can lead to inconsistencies.

5. Game Development:

Definition:

Android game development focuses on creating interactive and visually appealing games for Android devices.

Key Components:

Game engines like Unity, Unreal Engine.
Specialized programming languages (e.g., C# for Unity).

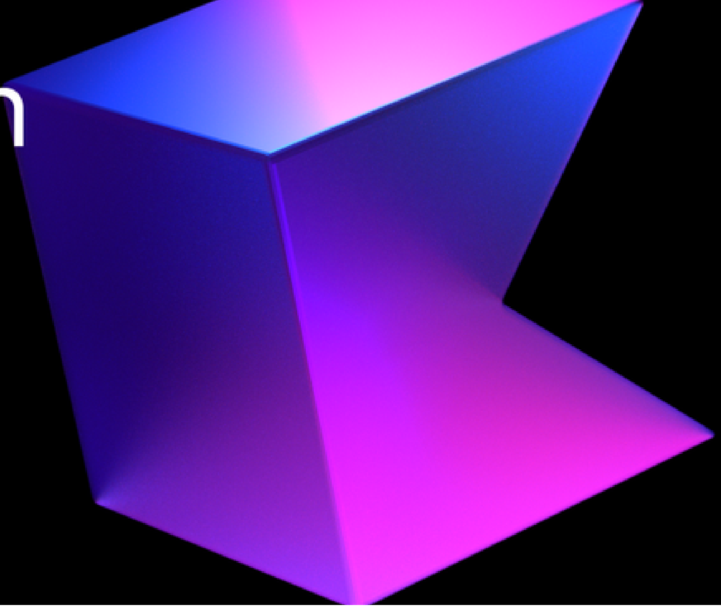
Advantages:

Powerful graphics and multimedia capabilities.
Support for virtual reality (VR) and augmented reality (AR) gaming.

Considerations:

Steeper learning curve for complex game development.
Specific skills required for game design and optimization.

Introduction to Kotlin



Hello, world!

```
fun main(args: Array<String>) {
println("Hello, world!")
}
```

```
fun main() {
println("Hello, world!")
}
```

```
fun main() = println("Hello, world!")
```

Where is ";"???

The basics

```
fun main(args: Array<String>) {
print("Hello")
println(", world!")
}
```

- An entry point of a Kotlin application is the main **top-level** function.
- It accepts a variable number of String arguments that can be omitted.
- print prints its argument to the standard output.
- println prints its arguments and adds a line break.

Variables

`val/var myValue: Type = someValue`

- var - mutable
- val - immutable
- Type can be inferred in most cases
- Assignment can be deferred

```
val a: Int = 1  // immediate assignment

var b = 2       // 'Int' type is inferred
b = a           // Reassigning to 'var' is okay

val c: Int      // Type required when no initializer is provided
c = 3           // Deferred assignment
a = 4           // Error: Val cannot be reassigned
```

`const val/val myValue: Type = someValue`

- const val - compile-time const value
- val - immutable value
- for const val use uppercase for naming

```
const val NAME = "Kotlin"   // can be calculated at compile-time
val nameLowered = NAME.lowercase()   // cannot be calculated at compile-
time
```

Functions

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

fun mul(a: Int, b: Int) = a * b

fun printMul(a: Int, b: Int): Unit {
    println(mul(a, b))
}

fun printMul1(a: Int = 1, b: Int) {
    println(mul(a, b))
}

fun printMul2(a: Int, b: Int = 1) = println(mul(a, b))
```

Single expression function.
Unit means that the function does not return anything meaningful.
It can be omitted.
Arguments can have **default** values.

If expression

```
fun maxOf(a: Int, b: Int): Int {
    if (a > b) {
        return a
    } else {
        return b
    }
}
```

is the same as $\longrightarrow$

```
fun maxOf(a: Int, b: Int) =
if (a > b) {
    a
} else {
    b
}
```

if can be an expression (it can return).

Can be a one-liner:

```
fun maxOf(a: Int, b: Int) = if (a > b) a else b
```

When expression

```
when (x) {
    1 -> print("x == 1")
    2 -> print("x == 2")
    else -> {
        print("x is neither 1 nor 2")
    }
}
```

when returns, the same way that if does.

```
when {
    x < 0 -> print("x < 0")
    x > 0 -> print("x > 0")
    else -> {
        print("x == 0")
    }
}
```

The condition can be inside of the branches.

```
fun serveTeaTo(customer: Customer) {
    val teaSack = takeRandomTeaSack()

when (teaSack) {
    is OolongSack -> error("We don't serve Chinese tea like $teaSack!")
    in trialTeaSacks, teaSackBoughtLastNight ->
error("Are you insane?! We cannot serve uncertified tea!")
}

teaPackage.brew().serveTo(customer)
}
```

when can accept several options in one branch. else branch can be omitted if when block is used as a *statement*.

&& vs and

`if (a && b) { ... }`   VS   `if (a and b) { ... }`
Unlike the && operator, this function does not perform short-circuit evaluation.

The same behavior with OR:
`if (a || b) { ... }`   VS   `if (a or b) { ... }`

Loops

```
val items = listOf("apple", "banana", "kiwifruit")

for (item in items) {
    println(item)
}

for (index in items.indices) {
    println("item at $index is ${items[index]}")
}

for ((index, item) in items.withIndex()) {
    println("item at $index is $item")
}
```

There are break and continue labels for loops:

```
myLabel@ for (item in items) {
    for (anotherItem in otherItems) {
        if (...) break@myLabel
        else continue@myLabel
    }
}
```

Ranges

```
val x = 10
if (x in 1..10) {
    println("fits in range")
}

for (x in 1..5) {
    print(x)
}

for (x in 9 downTo 0 step 3) {
    print(x)
}
```

downTo and step are extension functions, not keywords.
`'..'` is actually `T.rangeTo(that: T)`

Null safety

```
val notNullText: String = "Definitely not null"
val nullableText1: String? = "Might be null"
val nullableText2: String? = null

fun funny(text: String?) {
    if (text != null)
        println(text)
    else
        println("Nothing to print :(")
}

fun funnier(text: String?) {
    val toPrint = text ?: "Nothing to print :("
    println(toPrint)
}
```

Elvis operator ?:

If the expression to the left of ?: is not null, the Elvis operator returns it; otherwise, it returns the expression to the right.
Note that the expression on the right-hand side is evaluated only if the left-hand side is null.

```
fun loadInfoById(id: String): String? {
    val item = findItem(id) ?: return null
    return item.loadInfo() ?: throw Exception("...")
}
```

?:-)

Safe Calls

`someThing?.otherThing` does not throw an NPE if `someThing` is `null`.

Safe calls are useful in chains. For example, an employee may be assigned to a department (or not). That department may in turn have another employee as a department head, who may or may not have a name, which we want to print:

```
fun printDepartmentHead(employee: Employee) {
    println(employee.department?.head?.name)
}
```

To print only for non-null values, you can use the safe call operator together with let:

```
employee.department?.head?.name?.let { println(it) }
```

Unsafe Calls

The not-null assertion operator (!!) converts any value to a non-null type and throws an NPE exception if the value is null.

```
fun printDepartmentHead(employee: Employee) {
    println(employee.department!!.head!!.name!!)
}
```

Please, avoid using unsafe calls!

String templates and the string builder

```
val i = 10
val s = "Kotlin"

println("i = $i")
println("Length of $s is ${s.length}")

val sb = StringBuilder()
sb.append("Hello")
sb.append(", world!")
println(sb.toString())
```

What is XML ?

XML stands for Extensible Markup Language. XML is a markup language much like HTML used to describe data. It is derived from Standard Generalized Markup Language(SGML). Basically, the XML tags are not predefined in XML. We need to implement and define the tags in XML. XML tags define the data and used to store and organize data. It's easily scalable and simple to develop. In Android, the XML is used to implement UI-related data, and it's a lightweight markup language that doesn't make layout heavy. XML only contains tags, while implementing they need to be just invoked.

The simple syntax of XML is

<tag_name>Hello World!</tag_name>

So in this article, there is a deep discussion on how to learn and understand what XML is for Android Development.

Basics of User Interface(UI)

Basically in Android XML is used to implement the UI-related data. So understanding the core part of the UI interface with respect to XML is important. The User Interface for an Android App is built as the hierarchy of main layouts, widgets. The layouts are ViewGroup objects or containers that control how the child view should be positioned on the screen. Widgets here are view objects, such as Buttons and text boxes. Considering the following simple example of the activity_main.xml file.

the basic building block for user interface is a View object that is created from the View class and occupies a rectangular area on the screen. Views are the base class for UI components like TextView, Button, EditText etc.

the ViewGroup is a subclass of View. One or more Views can be grouped together into a ViewGroup. A ViewGroup provides the android layout in which we can order the appearance and sequence of views. Examples of ViewGroup are LinearLayout, FrameLayout, RelativeLayout etc.

Android Layout Types

Android provides the following ViewGroups or layouts:

- LinearLayout : is a ViewGroup that aligns all children in a single direction, vertically or horizontally
- RelativeLayout : is a ViewGroup that displays child views in relative positions
- Absolute Layout : allows us to specify the exact location of the child views and widgets
- TableLayout : is a view that groups its child views into rows and columns
- FrameLayout : is a placeholder on screen that is used to display a single view

Android Layout Attributes

- android:id : This is the ID which uniquely identifies the view
- android:layout_width : This is the width of the layout
- android:layout_height : This is the height of the layout
- android:layout_margin : This is the extra space outside of the view. For example if you give android:marginLeft=20dp, then the view will be arranged after 20dp from left
- android:layout_padding : This is similar to android:layout_margin except that it specifies the extra space inside the view
- android:layout_gravity : This specifies how child Views are positioned
- android:layout_weight : This specifies how much of the extra space in the layout should be allocated to the view
- android:layout_x : This specifies the x-coordinate of the layout
- android:layout_y : This specifies the y-coordinate of the layout
- android:layout_width=wrap_content tells the view to size itself to the dimensions required by its content.
- android:layout_width=match_parent tells the view to become as big as its parent view.

View Identification

The syntax for an ID, inside an XML tag is:

- The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource
- The plus-symbol (+) means that this is a new resource name that must be created and added to our resources

Android LinearLayout

- Android LinearLayout organizes elements along a single line. We can specify whether that line is vertical or horizontal using android:orientation. The orientation is horizontal by default.
- A vertical LinearLayout will only have one child per row (so it is a column of single elements), and a horizontal LinearLayout will only have one single row of elements on the screen.
- android:layout_weight attribute depicts the importance of the element. An element with larger weight occupies more screen space. Here is a sample Layout XML using LinearLayout:
- layout_linear.xml

```
<LinearLayout xmlns:android="https://schemas.android.com/apk/res/android"
android:orientation="vertical" android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:layout_margin="@dimen/activity_horizontal_margin"> <Button
android:id="@+id/backbutton" android:text="Back" android:layout_width="wrap_content"
android:layout_height="wrap_content" /> <TextView android:text="Row 2"
android:layout_width="wrap_content" android:textSize="18sp"
android:layout_margin="10dp" android:layout_height="wrap_content" /> <TextView
android:text="Row 3" android:textSize="18sp" android:layout_margin="10dp"
android:layout_width="wrap_content" android:layout_height="wrap_content" /> <TextView
android:text="Row 4" android:textSize="18sp" android:layout_margin="10dp"
android:layout_width="wrap_content" android:layout_height="wrap_content" /> <TextView
android:text="Row 5" android:textSize="18sp" android:layout_margin="10dp"
android:layout_width="wrap_content" android:layout_height="wrap_content" />
<LinearLayout android:orientation="horizontal" android:layout_width="match_parent"
android:layout_height="wrap_content"> <Button android:id="@+id/next_button"
android:text="next" android:layout_width="wrap_content"
android:layout_height="wrap_content" /> <TextView android:text="Row 6b"
android:textSize="18sp" android:layout_margin="10dp" android:layout_gravity="center"
android:layout_width="wrap_content" android:layout_height="wrap_content" /> <TextView
android:text="Row 6c" android:textSize="18sp" android:layout_margin="10dp"
android:layout_gravity="center" android:layout_width="wrap_content"
android:layout_height="wrap_content" /> <TextView android:text="Row 6d"
android:textSize="18sp" android:layout_margin="10dp" android:layout_gravity="center"
android:layout_width="wrap_content" android:layout_height="wrap_content" />
</LinearLayout> </LinearLayout>
```

Android RelativeLayout

- Android RelativeLayout lays out elements based on their relationships with one another, and with the parent container. This is one of the most complicated layout and we need several properties to actually get the layout we desire.
- That is, using RelativeLayout we can position a view to be toLeftOf, toRightOf, below or above its siblings.
- We can also position a view with respect to its parent such as centered horizontally, vertically or both, or aligned with any of the edges of the parent RelativeLayout. If none of these attributes are specified on a child view then the view is by default rendered to the top left position.

Android RelativeLayout attributes

The following are the major attributes used across **RelativeLayout**. They lay across three different categories:

Relative To Container

- android:layout_alignParentBottom : Places the bottom of the element on the bottom of the container
- android:layout_alignParentLeft : Places the left of the element on the left side of the container
- android:layout_alignParentRight : Places the right of the element on the right side of the container
- android:layout_alignParentTop : Places the element at the top of the container
- android:layout_centerHorizontal : Centers the element horizontally within its parent container
- android:layout_centerInParent : Centers the element both horizontally and vertically within its container
- android:layout_centerVertical : Centers the element vertically within its parent container

Relative to Siblings

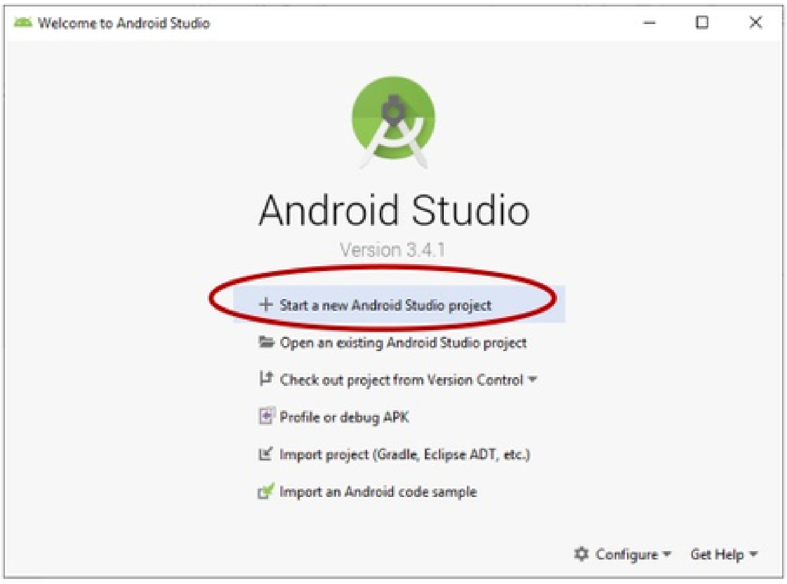
- android:layout_above : Places the element above the specified element
- android:layout_below : Places the element below the specified element
- android:layout_toLeftOf : Places the element to the left of the specified element
- android:layout_toRightOf : Places the element to the right of the specified element
- „@id/XXXXX” is used to reference an element by its id. One thing to remember is that referencing an element before it has been declared will produce an error so @+id/ should be used in such cases.

Alignment With Other Elements

- android:layout_alignBaseline : Aligns baseline of the new element with the baseline of the specified element
- android:layout_alignBottom : Aligns the bottom of new element in with the bottom of the specified element
- android:layout_alignLeft : Aligns left edge of the new element with the left edge of the specified element
- android:layout_alignRight : Aligns right edge of the new element with the right edge of the specified element
- android:layout_alignTop : Places top of the new element in alignment with the top of the specified element

```
<RelativeLayout android:layout_width="fill_parent" android:layout_height="fill_parent"
xmlns:android="https://schemas.android.com/apk/res/android"> <Button
android:id="@+id/backbutton" android:text="Back" android:layout_width="wrap_content"
android:layout_height="wrap_content" /> <TextView android:id="@+id/firstName"
android:text="First Name" android:textSize="18sp" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:layout_below="@id/backbutton" /> <TextView
android:id="@+id/editFirstName" android:text="JournalDev" android:textSize="18sp"
android:layout_width="wrap_content" android:layout_marginLeft="10dp"
android:layout_height="wrap_content" android:layout_toRightOf="@id/firstName"
android:layout_below="@id/backbutton"/> <TextView android:id="@+id/editLastName"
android:text="Layout Tutorial Example" android:textSize="18sp"
android:layout_width="wrap_content" android:layout_height="wrap_content"
android:layout_alignTop="@+id/lastName" android:layout_toRightOf="@+id/lastName"
android:layout_toEndOf="@+id/lastName" /> <TextView android:id="@+id/lastName"
android:text="Last Name" android:textSize="18sp" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:layout_marginTop="48dp"
android:layout_below="@+id/firstName" android:layout_alignParentLeft="true"
android:layout_alignParentStart="true" android:layout_marginRight="10dp"
android:layout_marginLeft="40dp" android:layout_marginStart="40dp" /> <Button
android:id="@+id/next" android:text="Next" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:layout_below="@+id/editLastName"
android:layout_alignLeft="@+id/editLastName" android:layout_alignStart="@+id/editLastName"
android:layout_marginTop="37dp" /> </RelativeLayout>
```

Click on Start a new Android Studio project

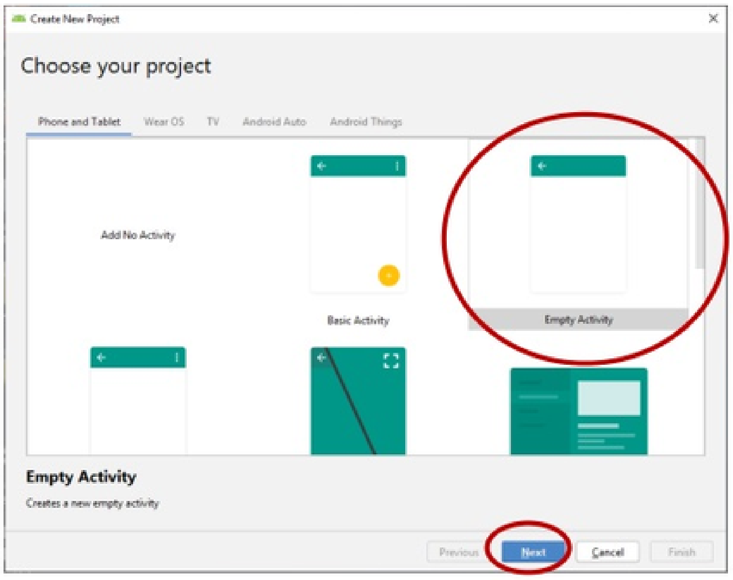

Welcome to Android Studio

Android Studio

Version 3.4.1

+ Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Profile or debug APK
Import project (Gradle, Eclipse ADT, etc.)
Import an Android code sample
Configure
Get Help

Choose an Empty Activity and click Next

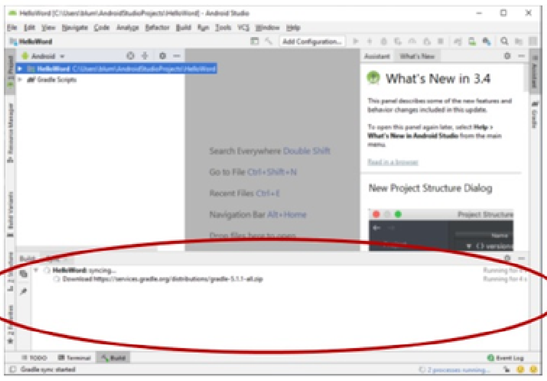

Create New Project

Choose your project

Empty Activity

Creates a new empty activity

Wait until the Sync-ing has taken place –
Sometimes it fails the first time and then re-syncs


HelloWord [C:\Users\blum\AndroidStudioProjects\HelloWord] - Android Studio

What's New in 3.4

This panel describes some of the new features and behavior changes included in this update.

To open this panel again later, select **Help > What's New in Android Studio** from the main menu.

Read in a browser

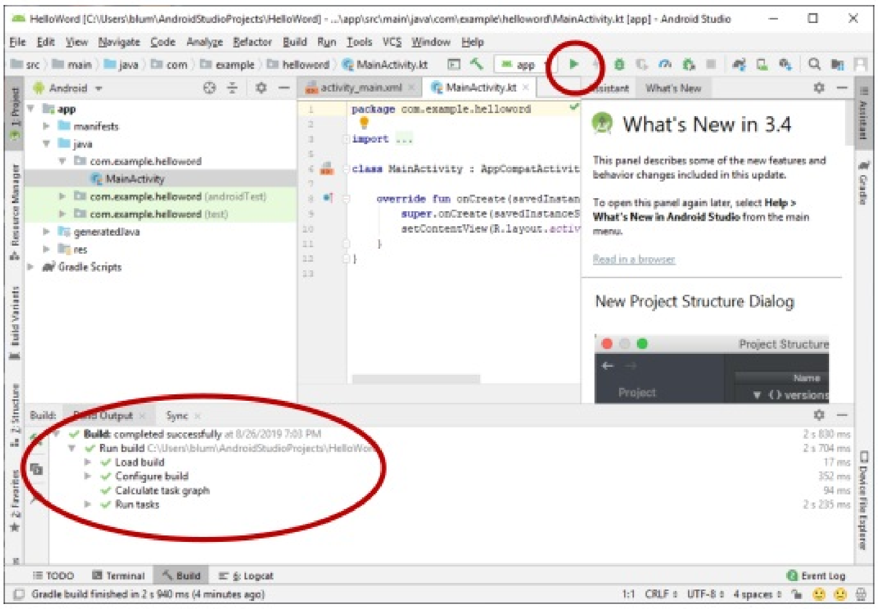
New Project Structure Dialog

```
HelloWord: syncing...
  Download https://services.gradle.org/distributions/gradle-5.1.1-all.zip
```

Gradle sync started

2 processes running...

When Sync area has green arrows, then click on Run arrow icon (or Run on menu)



HelloWord [C:\Users\blum\AndroidStudioProjects\HelloWord] - ...\app\src\main\java\com\example\helloword\MainActivity.kt [app] - Android Studio

```
package com.example.helloword

import ...

class MainActivity : AppCompatActivit

    override fun onCreate(savedInstan
        super.onCreate(savedInstanceS
        setContentView(R.layout.activ
    }
}
```

What's New in 3.4

This panel describes some of the new features and behavior changes included in this update.

To open this panel again later, select **Help > What's New in Android Studio** from the main menu.

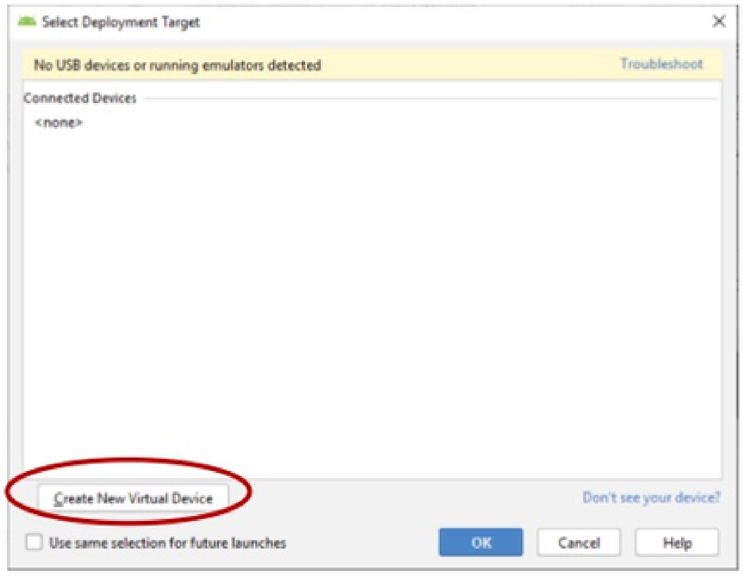
Read in a browser

New Project Structure Dialog

```
Build: completed successfully at 8/26/2019 7:03 PM      2 s 830 ms
  Run build C:\Users\blum\AndroidStudioProjects\HelloWo  2 s 704 ms
    Load build                                          17 ms
    Configure build                                     352 ms
    Calculate task graph                                94 ms
    Run tasks                                           2 s 235 ms
```

Gradle build finished in 2 s 940 ms (4 minutes ago)

Click on Create New Virtual Device button



Select Deployment Target

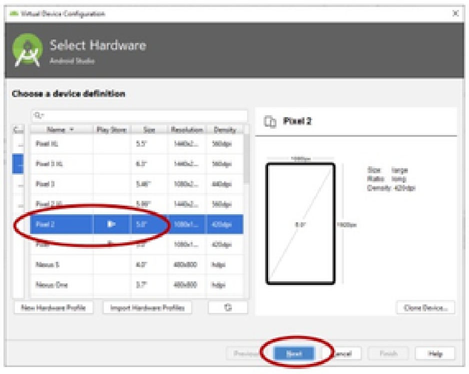
No USB devices or running emulators detected

Connected Devices

<none>

Troubleshoot
Create New Virtual Device
Don't see your device?
Use same selection for future launches
OK
Cancel
Help

I usually go with the default Pixel 2 choice and click Next



Select Hardware

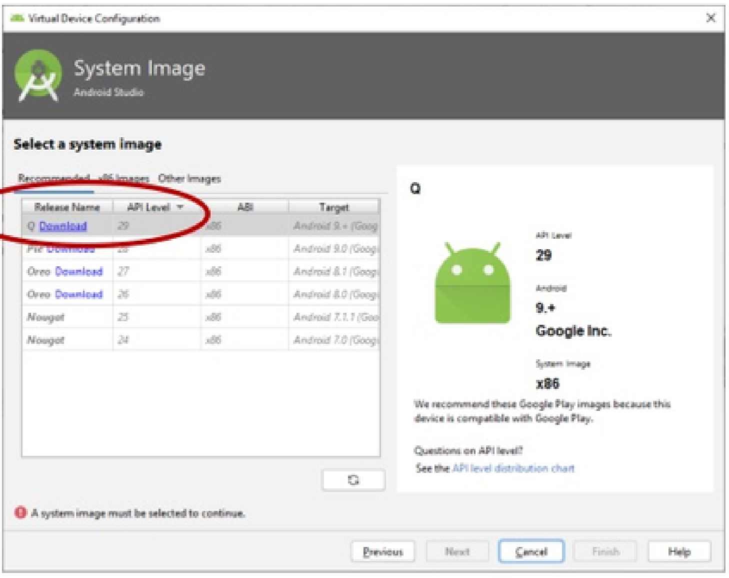
Choose a device definition

Name	Play Store	Size	Resolution	Density
Pixel XL		5.5"	1440x2...	560dpi
Pixel 3 XL		6.3"	1440x2...	560dpi
Pixel 3		5.46"	1080x2...	440dpi
Pixel 2 XL		5.99"	1440x2...	560dpi
Pixel 2	▶	5.0"	1080x1...	420dpi
Pixel	▶	5.0"	1080x1...	420dpi
Nexus 5		4.0"	480x800	hdpi
Nexus One		3.7"	480x800	hdpi

Pixel 2

Size: large
Ratio: long
Density: 420dpi

Choose a release Name (I go with Q) and click Download



Virtual Device Configuration

System Image
Android Studio

Select a system image

Recommended x86 Images Other Images

Release Name	API Level	ABI	Target
Q Download	29	x86	Android 9.+ (Goog
Pie Download	28	x86	Android 9.0 (Goog
Oreo Download	27	x86	Android 8.1 (Goog
Oreo Download	26	x86	Android 8.0 (Goog
Nougat	25	x86	Android 7.1.1 (Goo
Nougat	24	x86	Android 7.0 (Goog

Q

API Level
29

Android
9.+
Google Inc.

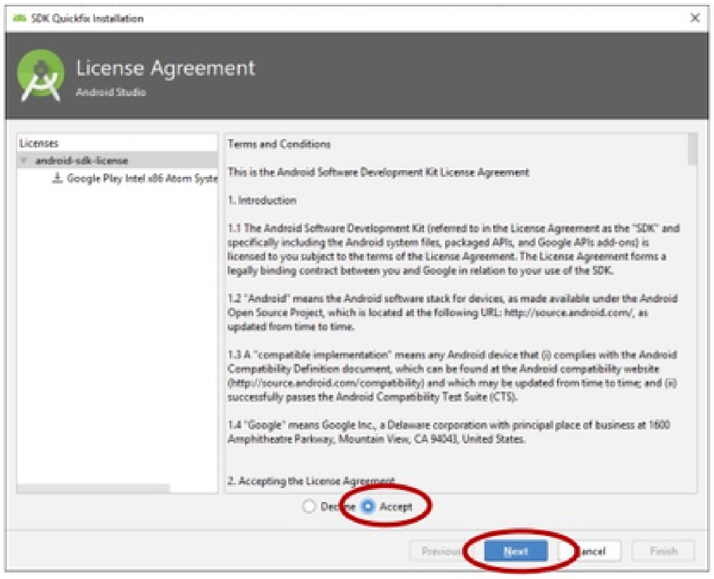
System Image
x86

We recommend these Google Play images because this device is compatible with Google Play.

Questions on API level?
See the API level distribution chart

A system image must be selected to continue.

Select Accept and click Next



SDK Quickfix Installation

License Agreement
Android Studio

Licenses
android-sdk-license
Google Play Intel x86 Atom Syste

Terms and Conditions

This is the Android Software Development Kit License Agreement

1. Introduction

1.1 The Android Software Development Kit (referred to in the License Agreement as the "SDK" and specifically including the Android system files, packaged APIs, and Google APIs add-ons) is licensed to you subject to the terms of the License Agreement. The License Agreement forms a legally binding contract between you and Google in relation to your use of the SDK.

1.2 "Android" means the Android software stack for devices, as made available under the Android Open Source Project, which is located at the following URL: http://source.android.com/, as updated from time to time.

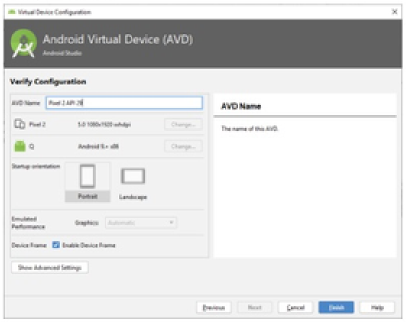
1.3 A "compatible implementation" means any Android device that (i) complies with the Android Compatibility Definition document, which can be found at the Android compatibility website (http://source.android.com/compatibility) and which may be updated from time to time; and (ii) successfully passes the Android Compatibility Test Suite (CTS).

1.4 "Google" means Google Inc., a Delaware corporation with principal place of business at 1600 Amphitheatre Parkway, Mountain View, CA 94043, United States.

2. Accepting the License Agreement

Decline [✓ Accept]

Click Finish



Virtual Device Configuration

Android Virtual Device (AVD)
Android Studio

Verify Configuration

AVD Name: Pixel 2 API 29

Pixel 2 5.0 1080x1920 xxhdpi Change...

Q Android 9.+ x86 Change...

Startup orientation: [✓ Portrait] Landscape

Emulated Performance: Graphics: Automatic

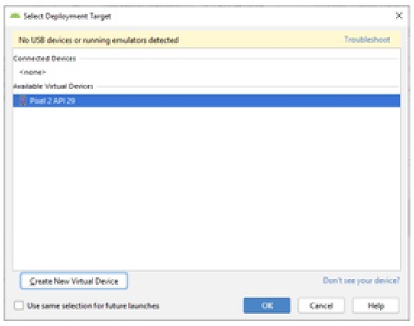
Device Frame: [✓] Enable Device Frame

Show Advanced Settings

AVD Name

The name of this AVD.

Click OK



Select Deployment Target

No USB devices or running emulators detected Troubleshoot

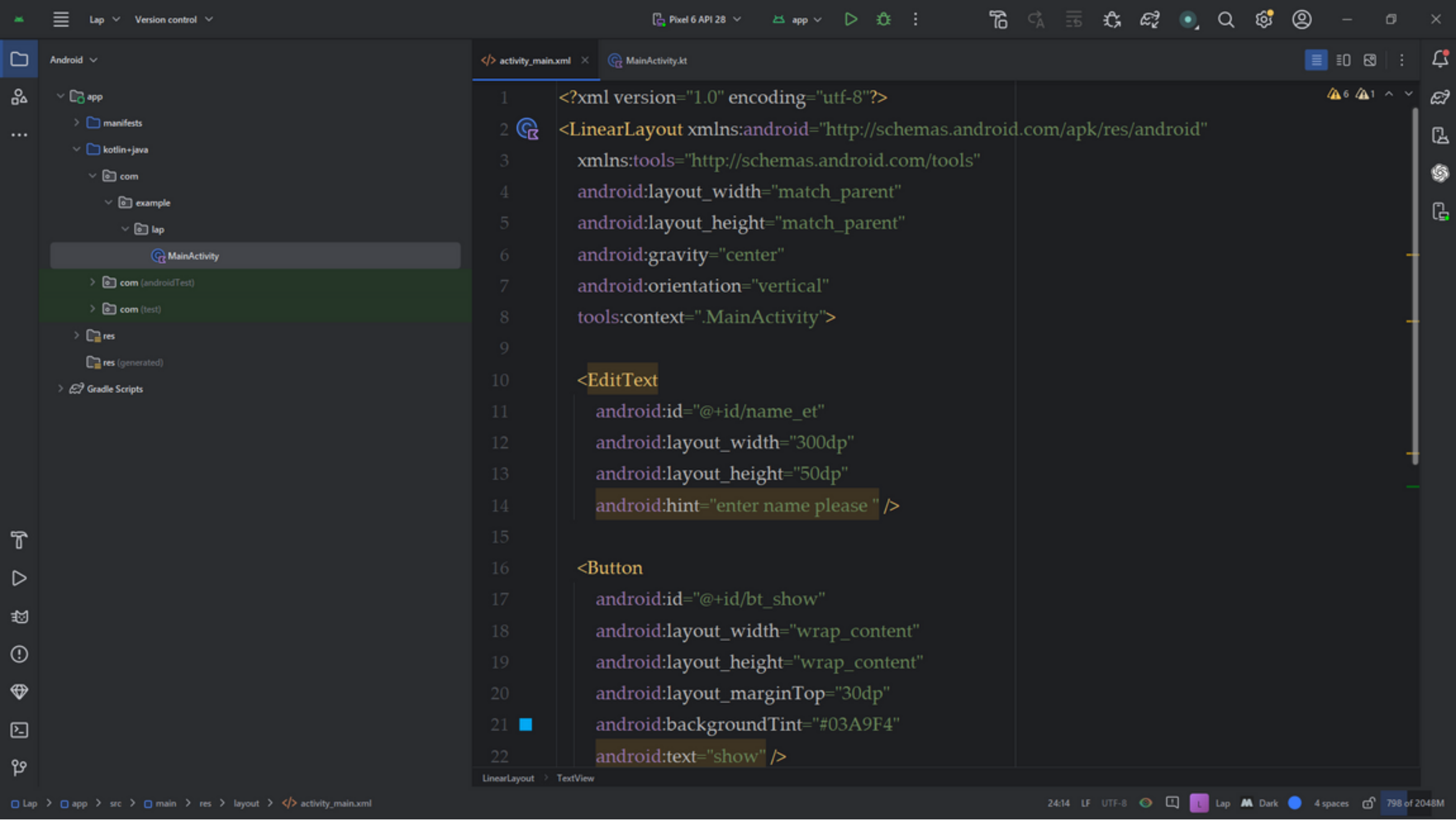
Connected Devices
<none>

Available Virtual Devices
Pixel 2 API 29

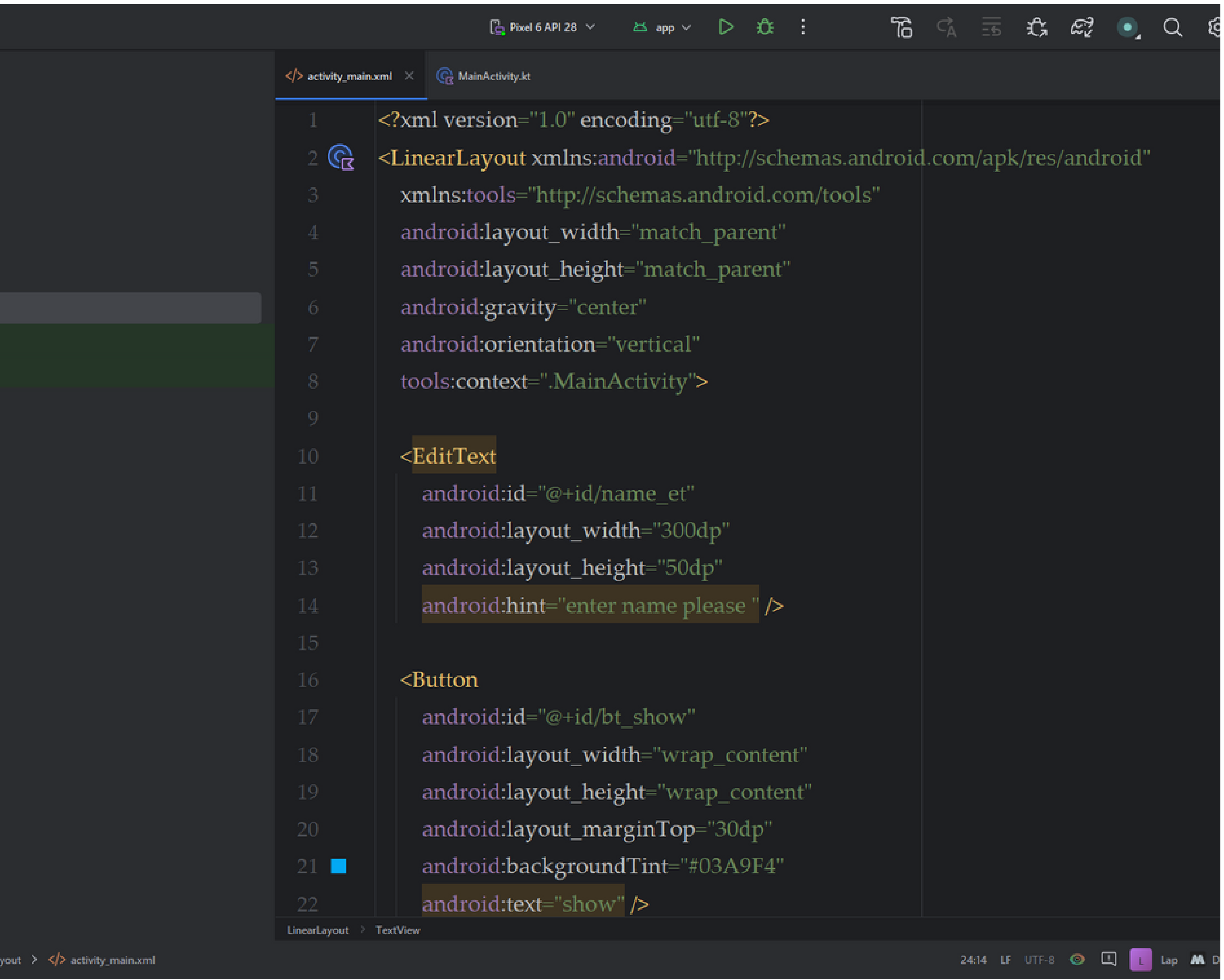
Create New Virtual Device

Don't see your device?

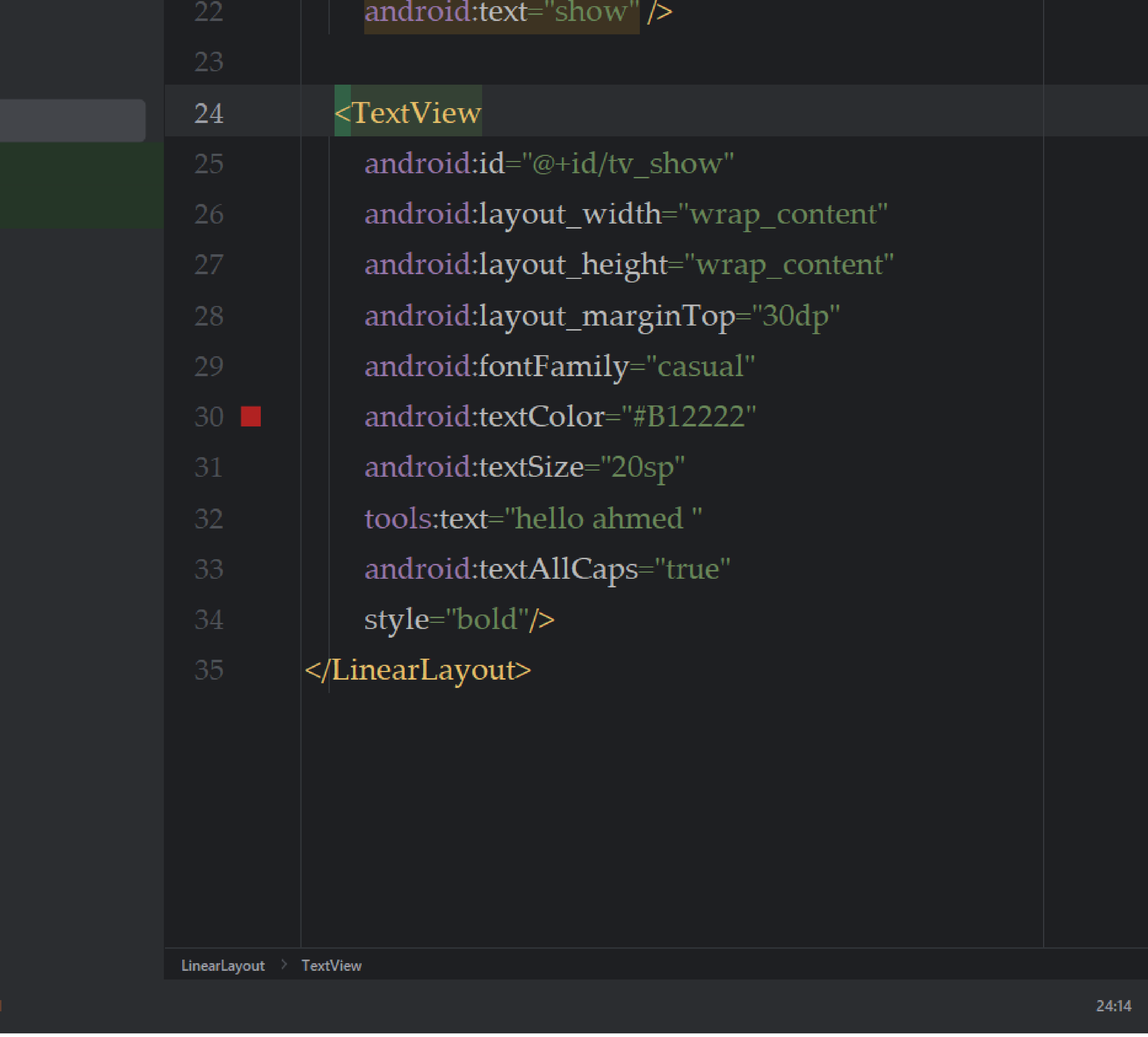
[ ] Use same selection for future launches



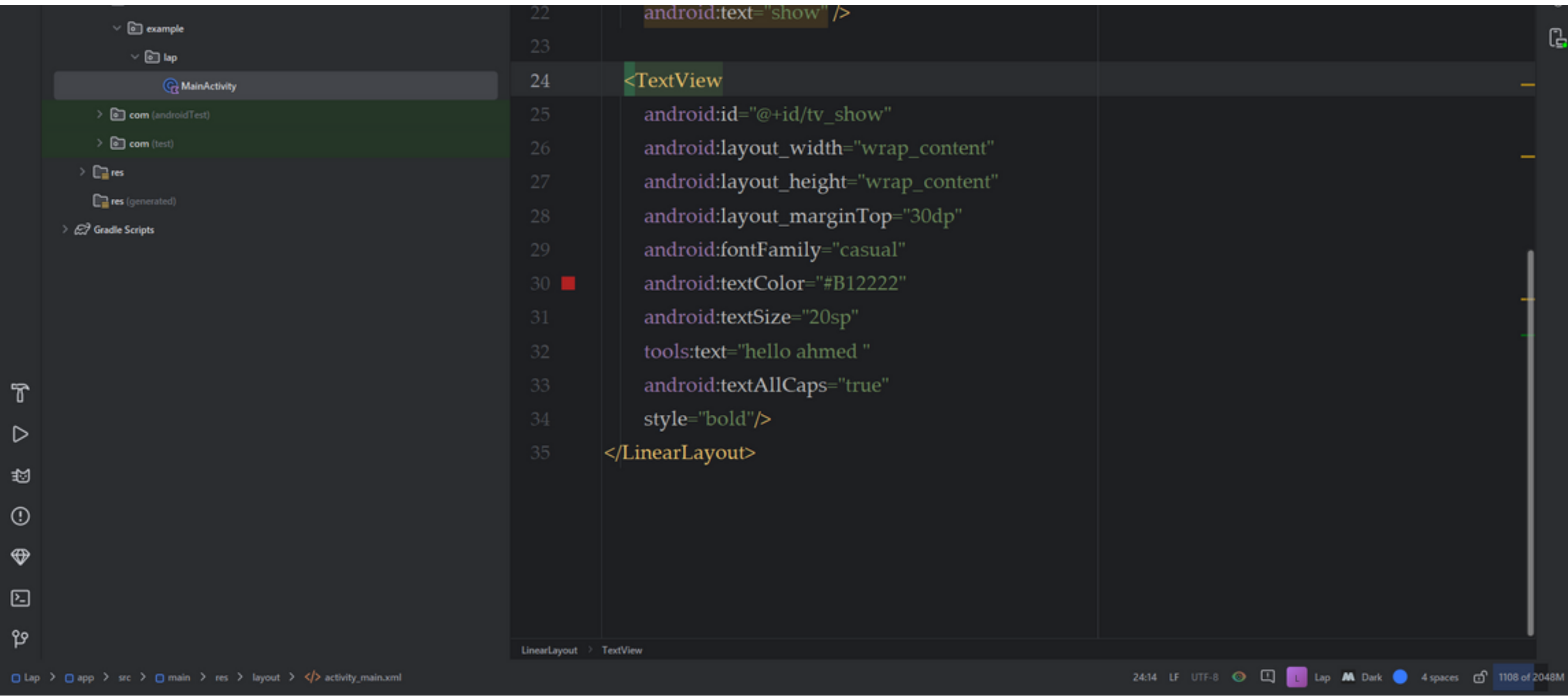
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:gravity="center"
  android:orientation="vertical"
  tools:context=".MainActivity">

  <EditText
    android:id="@+id/name_et"
    android:layout_width="300dp"
    android:layout_height="50dp"
    android:hint="enter name please " />

  <Button
    android:id="@+id/bt_show"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="30dp"
    android:backgroundTint="#03A9F4"
    android:text="show" />
```



```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:gravity="center"
  android:orientation="vertical"
  tools:context=".MainActivity">

  <EditText
    android:id="@+id/name_et"
    android:layout_width="300dp"
    android:layout_height="50dp"
    android:hint="enter name please " />

  <Button
    android:id="@+id/bt_show"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="30dp"
    android:backgroundTint="#03A9F4"
    android:text="show" />
```

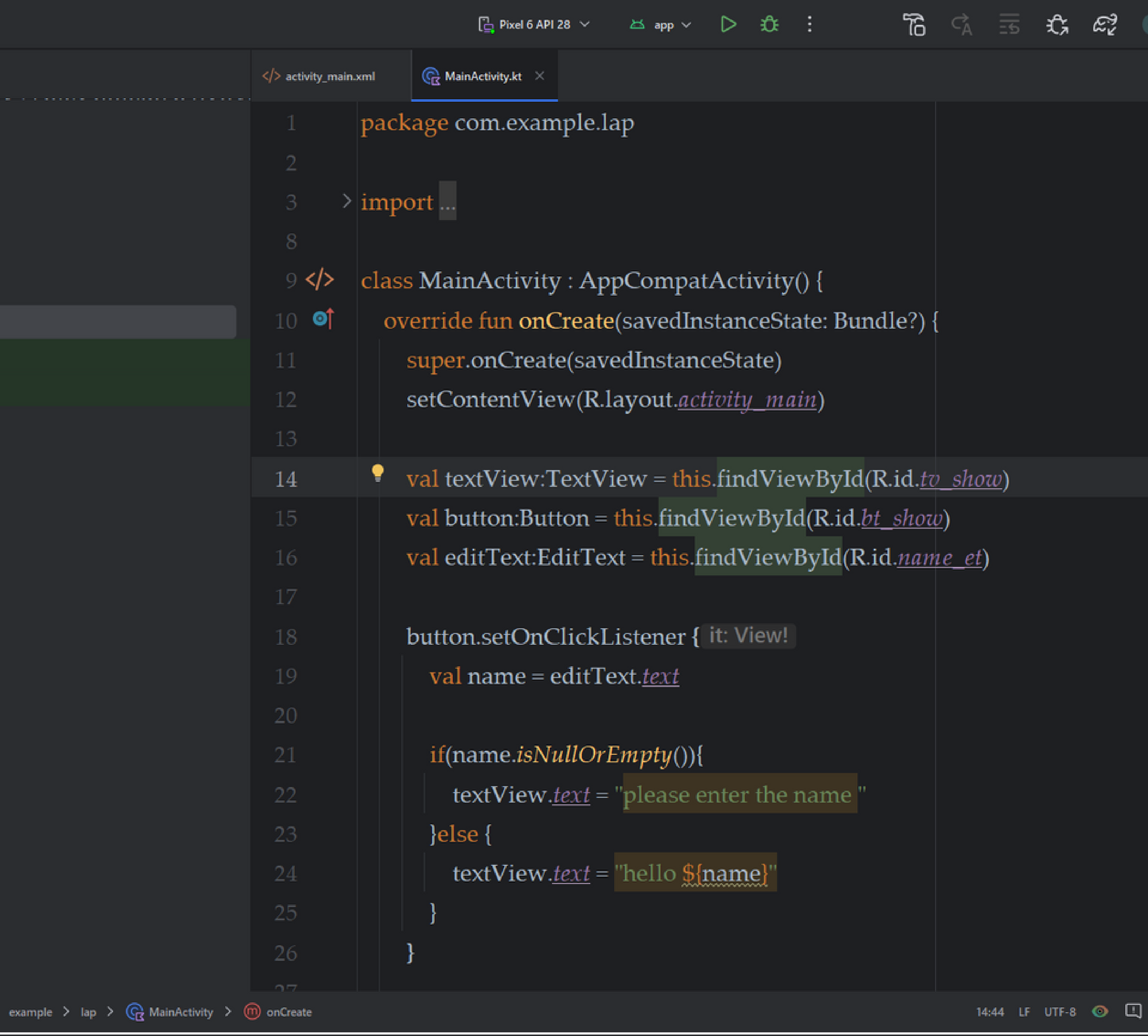


```xml
        android:text="show" />

    <TextView
        android:id="@+id/tv_show"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="30dp"
        android:fontFamily="casual"
        android:textColor="#B12222"
        android:textSize="20sp"
        tools:text="hello ahmed "
        android:textAllCaps="true"
        style="bold"/>
</LinearLayout>
```

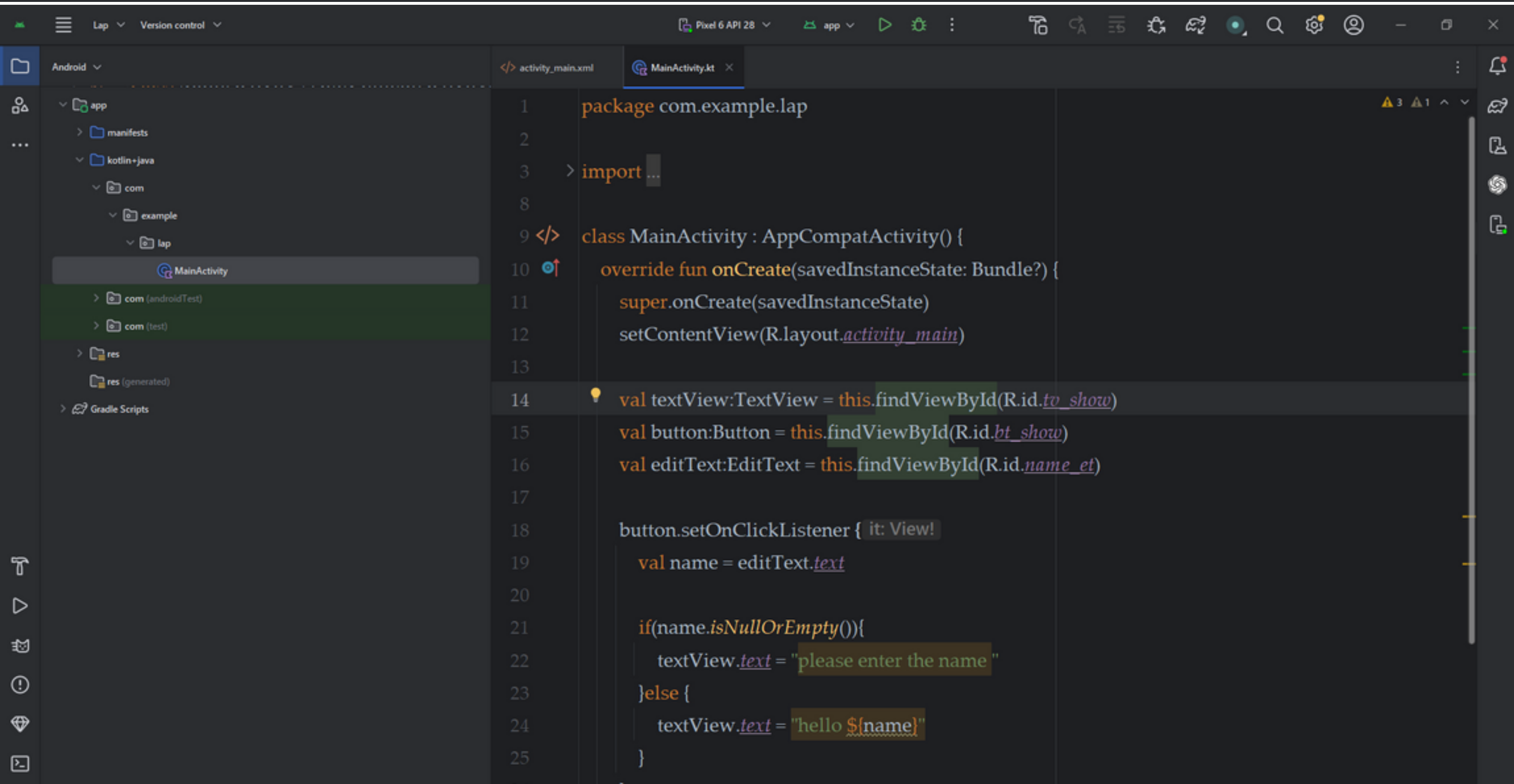


```xml
        android:text="show" />

    <TextView
        android:id="@+id/tv_show"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="30dp"
        android:fontFamily="casual"
        android:textColor="#B12222"
        android:textSize="20sp"
        tools:text="hello ahmed "
        android:textAllCaps="true"
        style="bold"/>
</LinearLayout>
```

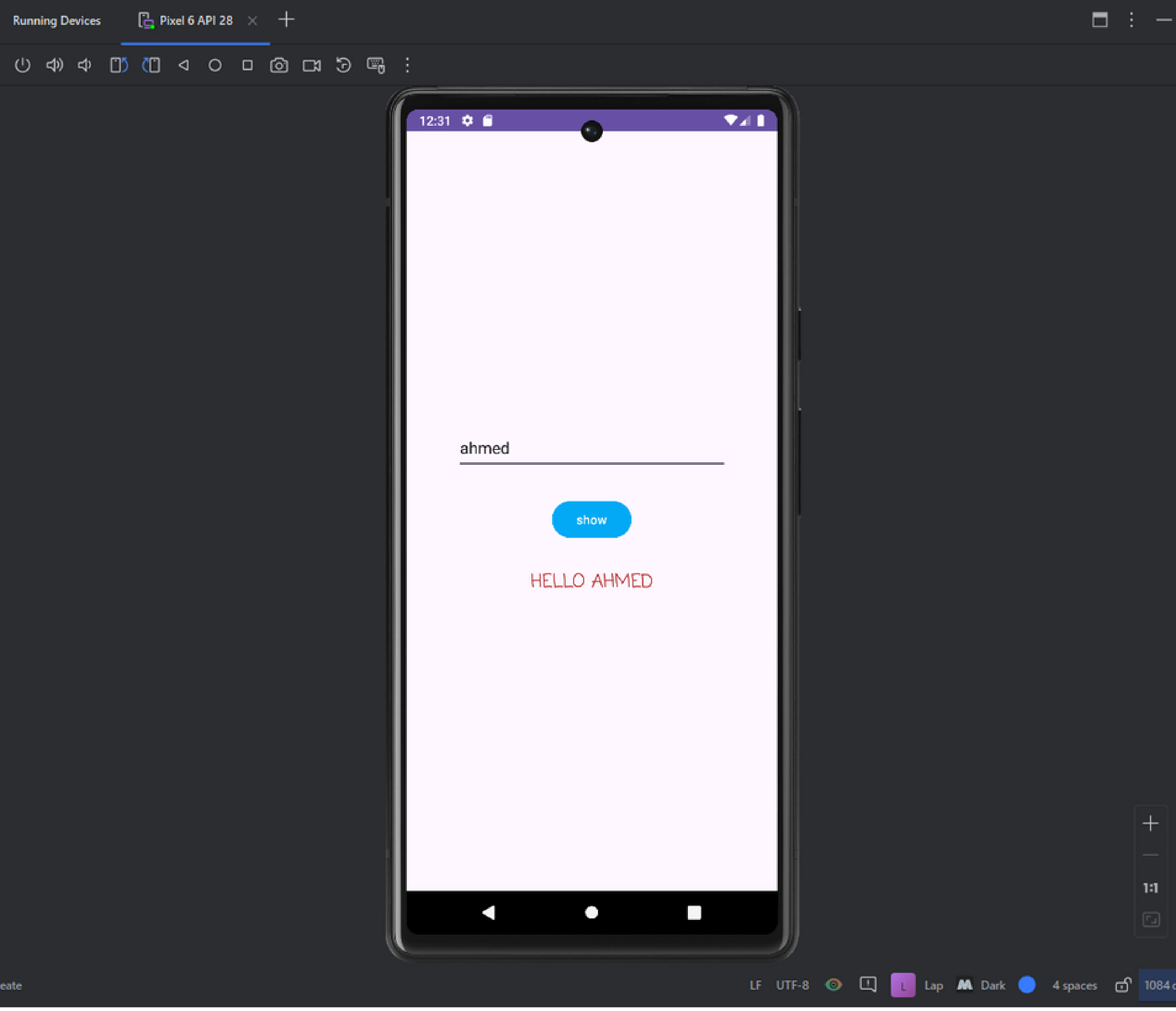
example
lap
MainActivity
com (androidTest)
com (test)
res
res (generated)
Gradle Scripts
LinearLayout
TextView
Lap > app > src > main > res > layout > activity_main.xml
24:14 LF UTF-8 Lap Dark 4 spaces 1108 of 2048M



```kotlin
package com.example.lap

import ...

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val textView:TextView = this.findViewById(R.id.tv_show)
        val button:Button = this.findViewById(R.id.bt_show)
        val editText:EditText = this.findViewById(R.id.name_et)

        button.setOnClickListener { it: View!
            val name = editText.text

            if(name.isNullOrEmpty()){
                textView.text = "please enter the name "
            }else {
                textView.text = "hello ${name}"
            }
        }
```



```kotlin
package com.example.lap

import ...

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val textView:TextView = this.findViewById(R.id.tv_show)
        val button:Button = this.findViewById(R.id.bt_show)
        val editText:EditText = this.findViewById(R.id.name_et)

        button.setOnClickListener { it: View!
            val name = editText.text

            if(name.isNullOrEmpty()){
                textView.text = "please enter the name "
            }else {
                textView.text = "hello ${name}"
            }
```


Running Devices
Pixel 6 API 28
12:31
ahmed
show
HELLO AHMED
1:1
eate
LF
UTF-8
Lap
Dark
4 spaces
1084 c