CS 473 - MDP
Mobile Device Programming

Maharishi International University

CS 473 - MDP
Mobile Device Programming

MS.CS Program

Department of Computer Science

Renuka Mohanraj , Ph.D.

Maharishi International
University

CS 473 – MDP
Mobile Device Programming

Lesson 3
Creating First App

Maharishi International University

Wholeness

- In this lecture we examine fundamental building blocks of Android apps including the manifest, the activity, and views. We also explore the useful files in your project explorer. Lastly, we create an Android app from these basic elements and run through Emulators. *The most fundamental knowledge is the most important knowledge, since everything else is built upon it. The reason TM can provide such a wide range of benefits to life in general is because it works at such a truly fundamental level.*

Agenda

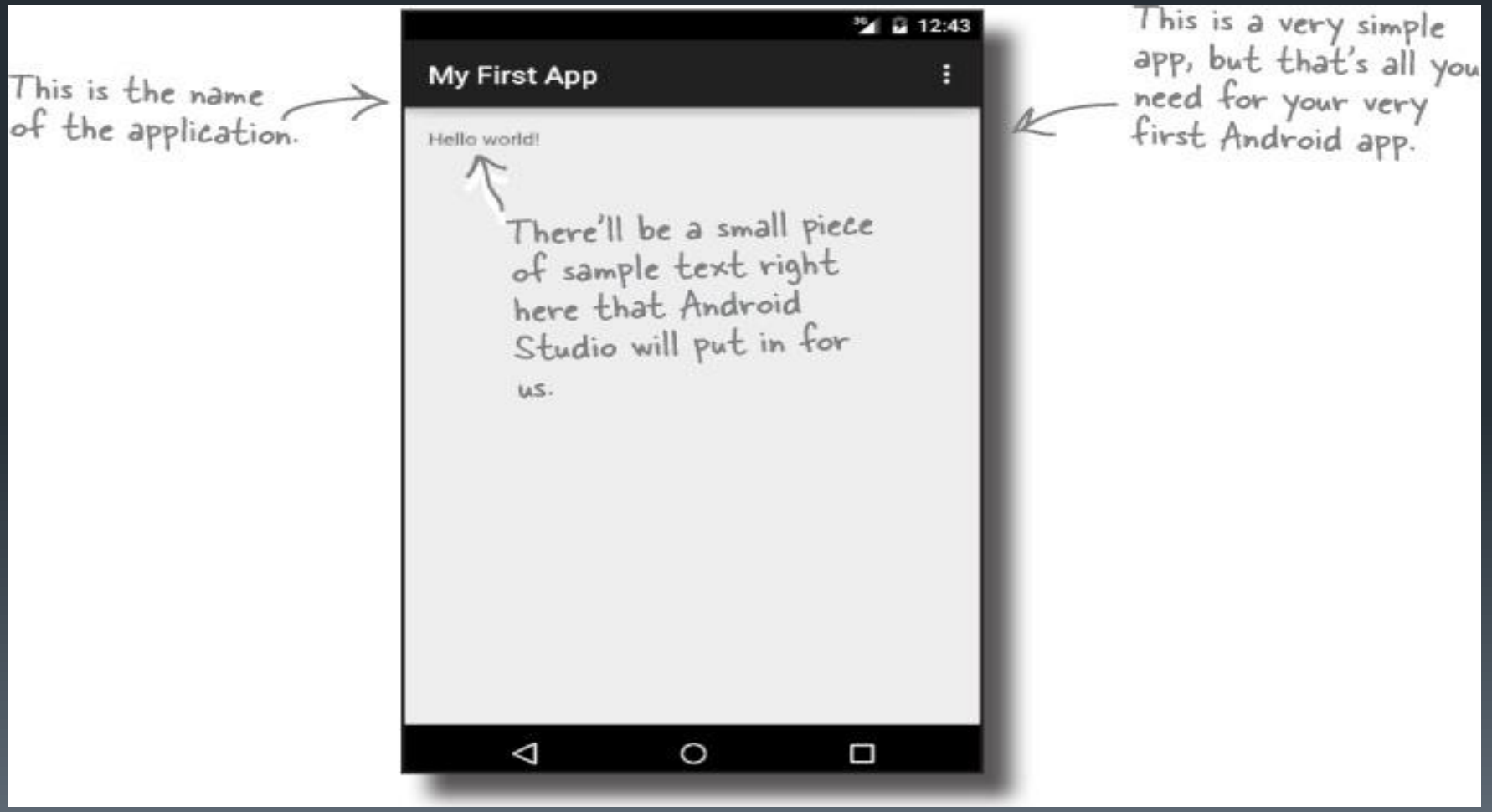
- Introduction to Android Studio
- Creation of HelloWorld app
- Android Project Structure and Folders
- AVD Manager
- Styles
- Event Handling

Introduction to Android Studio

- Android Studio is the new foundation of Google's efforts to give Android developers top-notch development tools.
- Android Studio is a special version of IntelliJ IDEA that interfaces with the Android Software Development Kit (SDK) and the gradle build system.
- Gradle is the native build system for Android Studio. Hence, if you are using that IDE, you should get a build.gradle file automatically. Will discuss about this in later chapters.

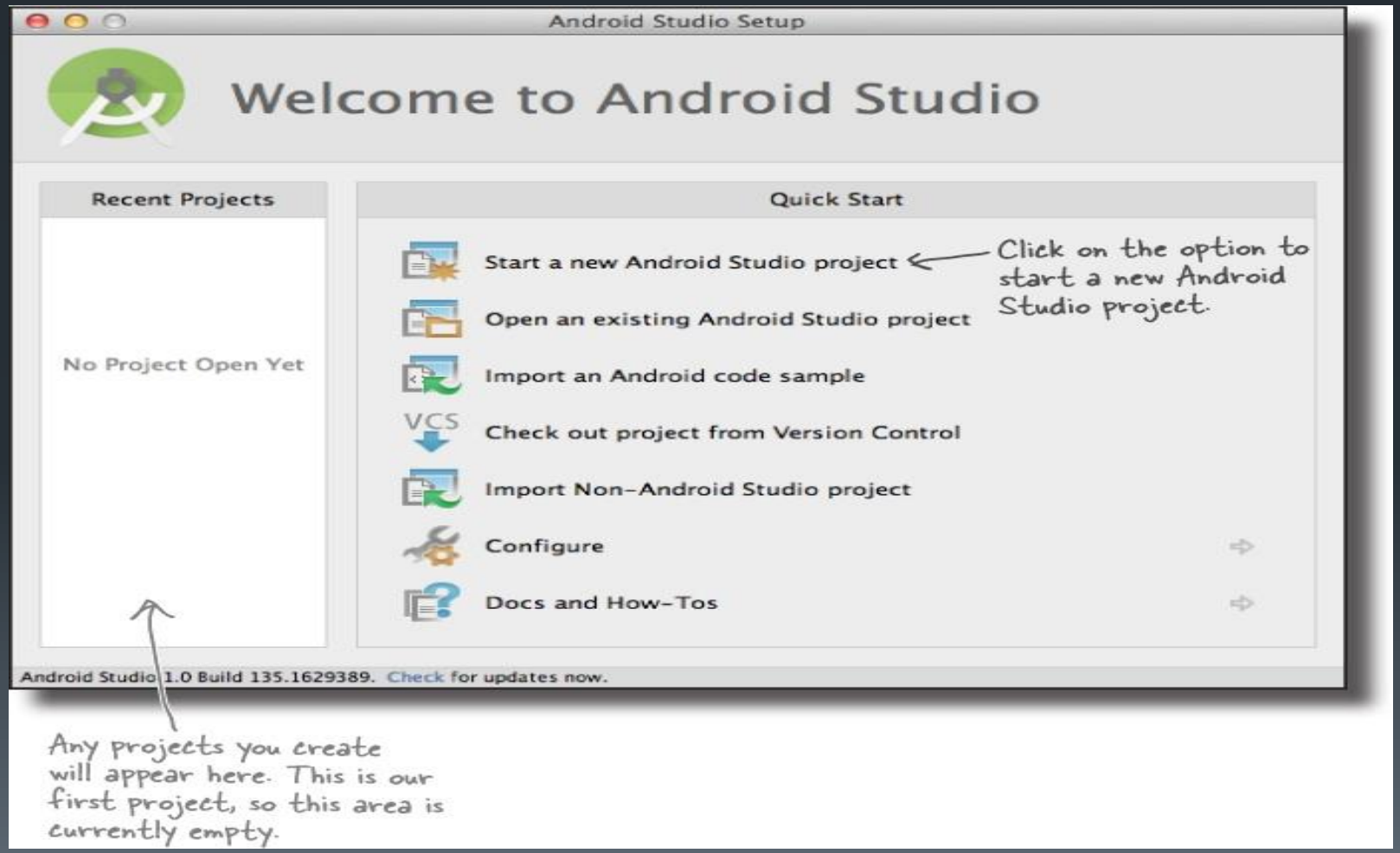
Initial HelloWorld App Example

- Now that you've set up your development environment, you're ready to create your first Android app. Here's what the app will look like:

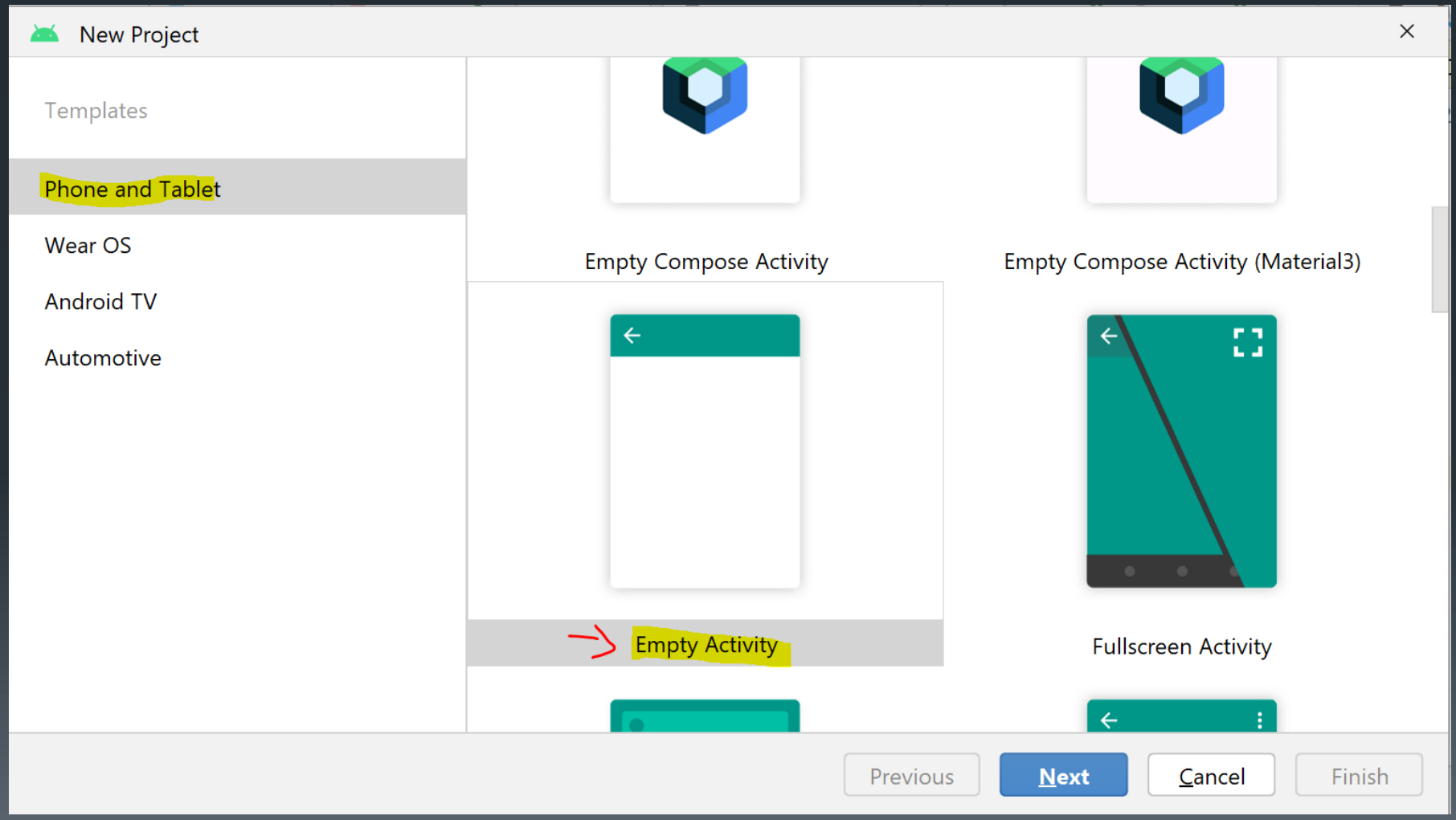


1. Create a new project

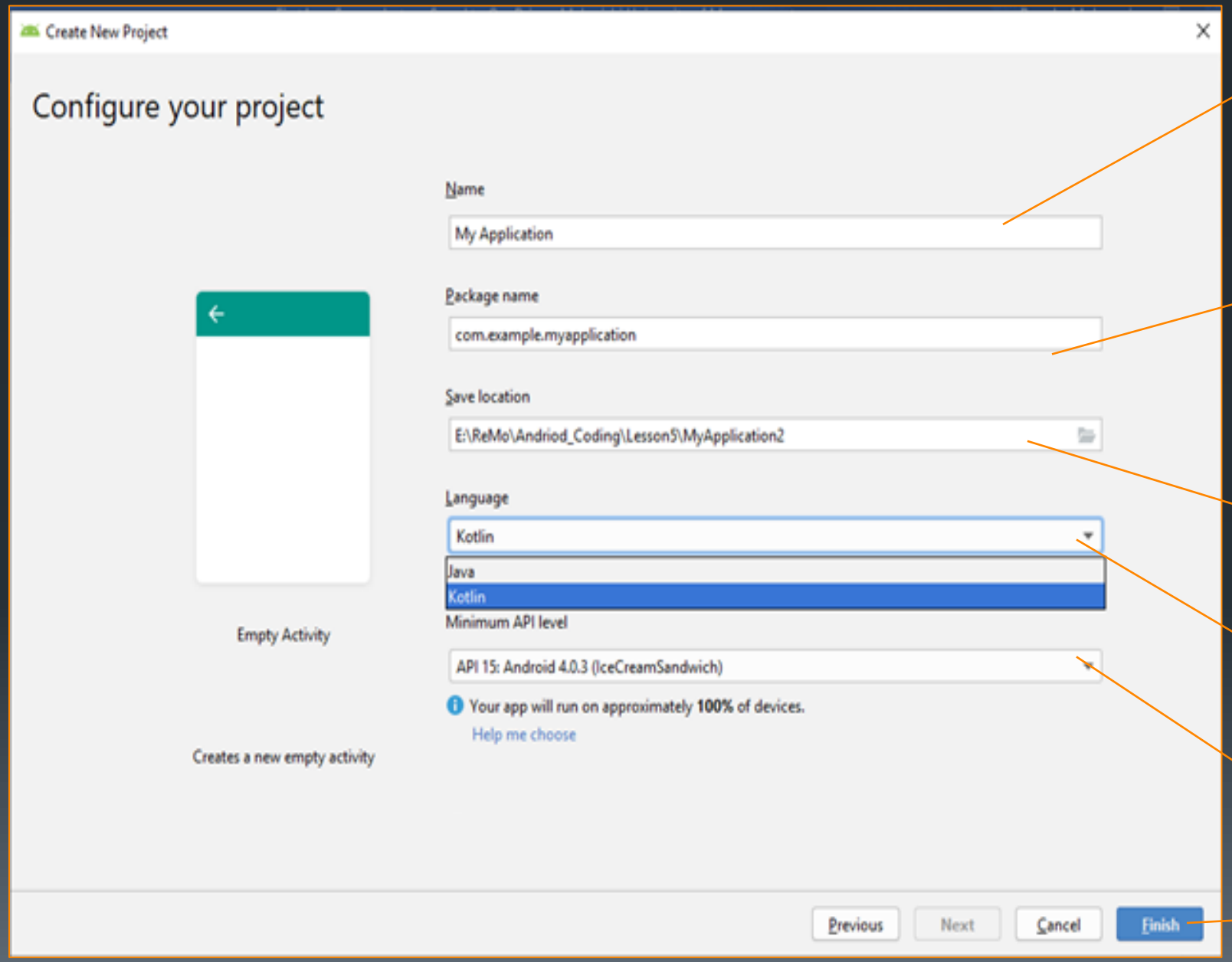
- The Android Studio welcome screen gives you several options for what you want to do. We want to create a new project, so click on the option for “Start a new Android Studio project”. (or) File -> New->New Project



Choose Phone and Tablets then select Empty Activity and click Next to proceed.



Create a new project



Name of your Application shown in Google play store and other places.

Forms the package name by combining application name and company name

All your files for your project will be stored here.

Choose the language for Implementation

Choose Minimum API support of SDK

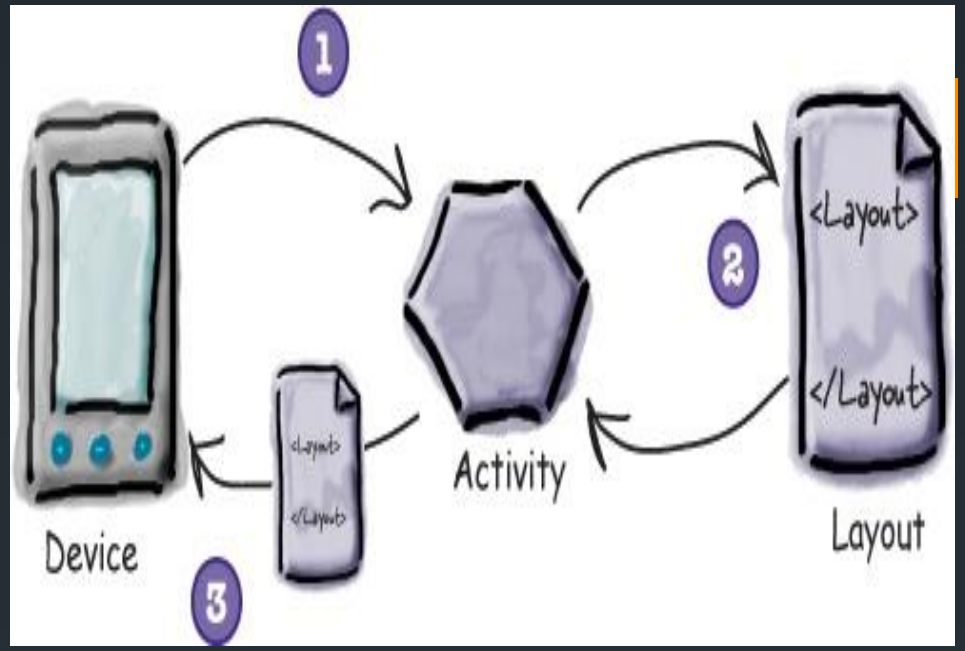
Click Finish

Activity

- Every Android app is a collection of screens, and each screen is comprised of **an activity** and a layout.
- An **activity** is **a single, defined thing that your user can do**. You might have an activity to compose an email, take a photo, or find a contact. Activities are usually associated with one screen, and they're written in Java/Kotlin.
- A **layout** describes **the appearance of the screen.** Layouts are written as XML files and they tell Android how the different screen elements are arranged.

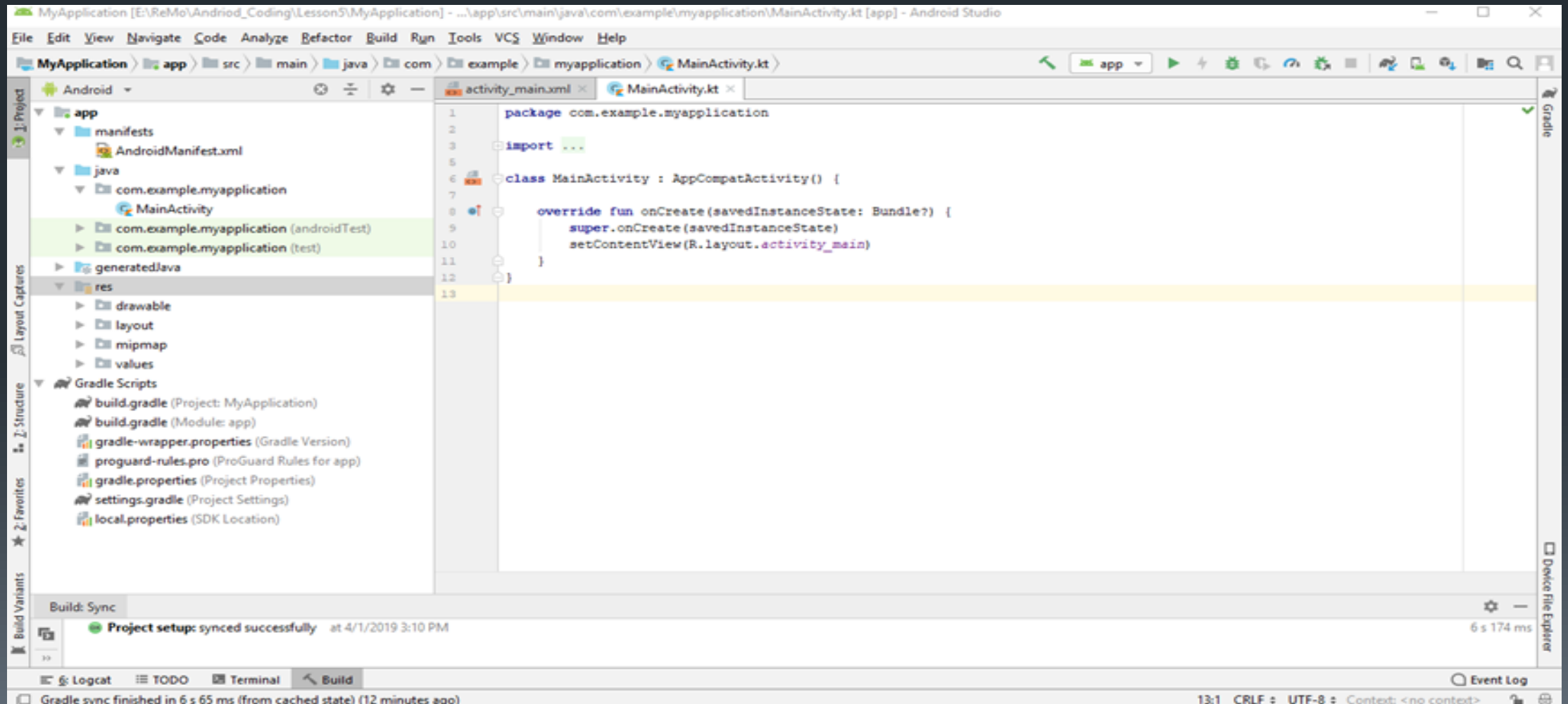
Layout & Activity

- ***Layouts define how the user interface is presented. Activities define actions.***
  1. **The device launches your app and creates an activity object.**
  2. **The activity object specifies a layout.**
  3. **The activity tells Android to display the layout on screen.**
  4. **The user interacts with the layout that's displayed on the device.**
  5. **The activity responds to these interactions by running application code.**
  6. **The activity updates the display...**
  7. **...which the user sees on the device.**



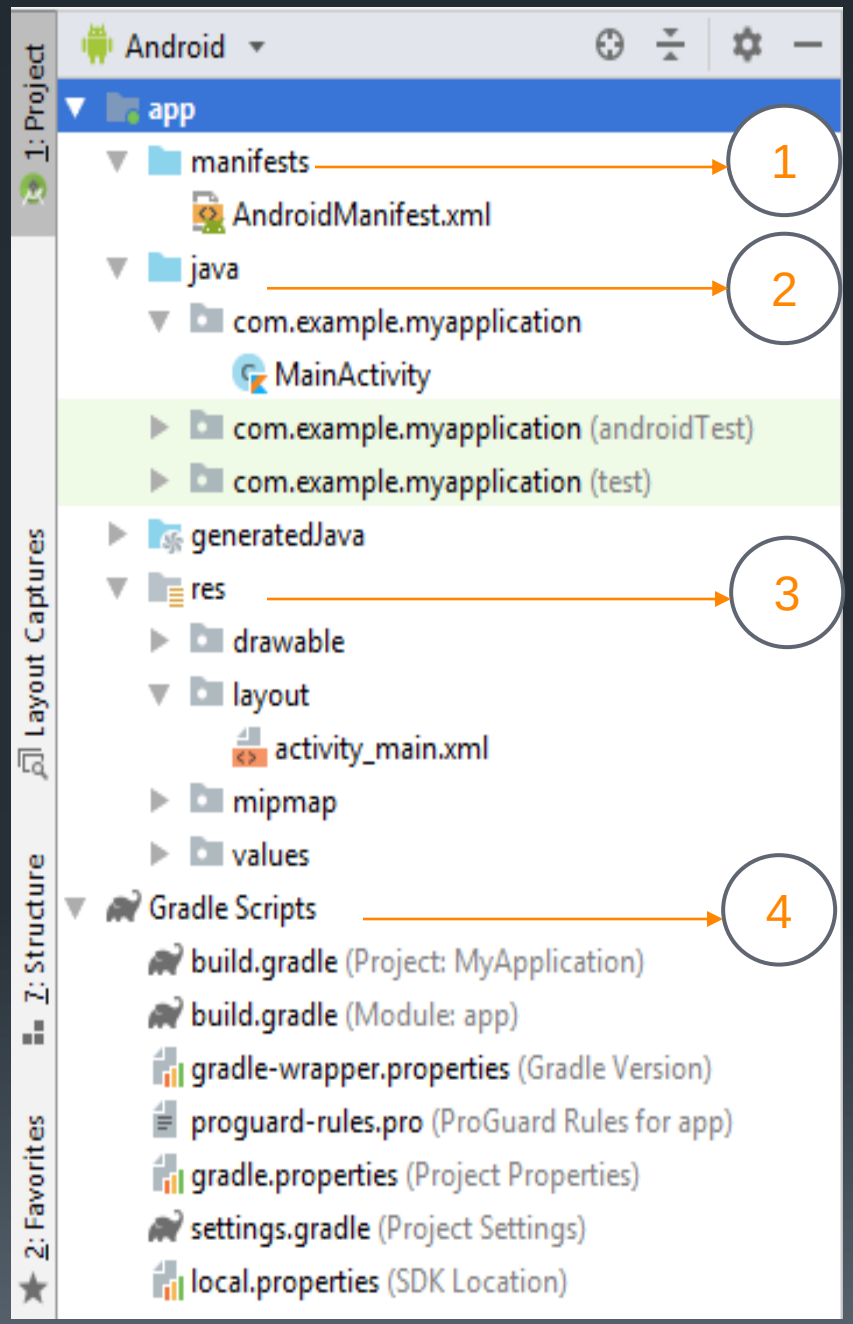
Creation of HelloWorldApp

- Here's what our project looks like (don't worry if it looks complicated right now, we'll break it down over the next few slides):
- Refer : https://kotlinlang.org/docs/tutorials/kotlin-android.html



Project folders

1. **manifests—**This folder contains AndroidManifest.xml**.** This file describes all the components of your Android app and is read by the Android runtime system when your app is executed.
2. **java—**All your Kotlin and Java language files are organized here.
3. **res—**This folder contains all the resources for your app, including images, layout files, strings, icons, and styling.
4. **Gradle Scripts -** Shows all the project's build-related configuration files.



Manifest.xml

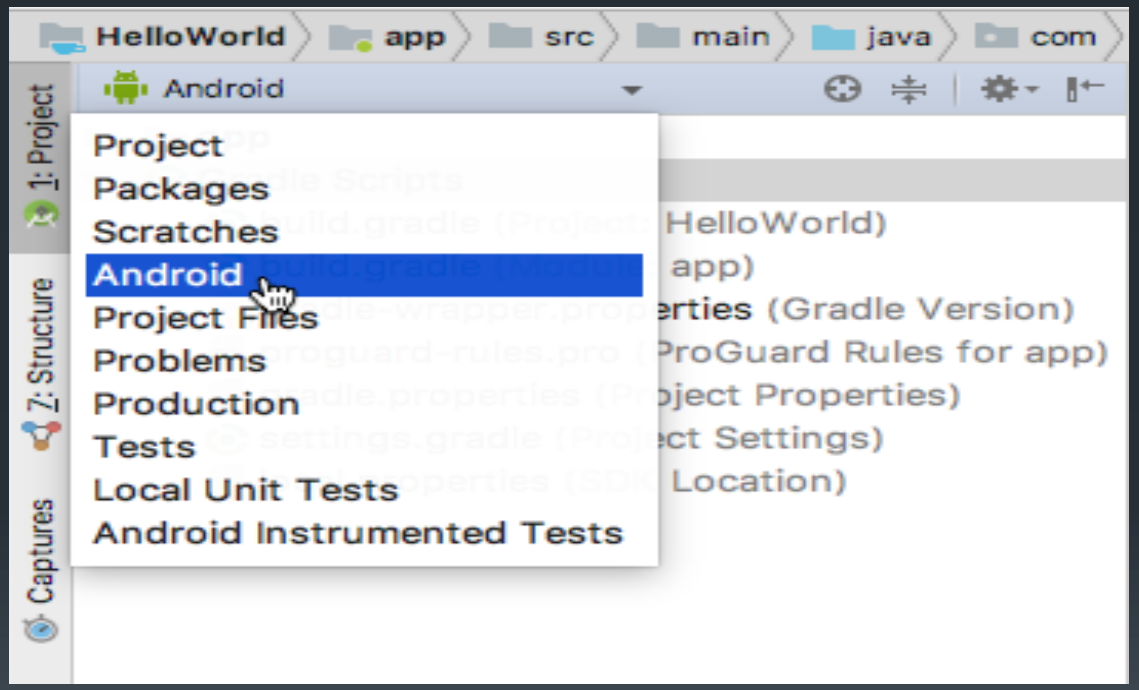
When system starts app. Manifest helps system to know all the components that exits in a app.
The components are:-

- **User Permissions**: Internet, Camera, Database, Phone contacts access etc.
- Declaring **Activities, Services, Content Provider** etc.
- Declaring **API Level**
- Declare Hardware and Software used by the App for example Camera, Bluetooth or multi touchscreen

In Short **Manifest** is the **SUMMARY** of your App.

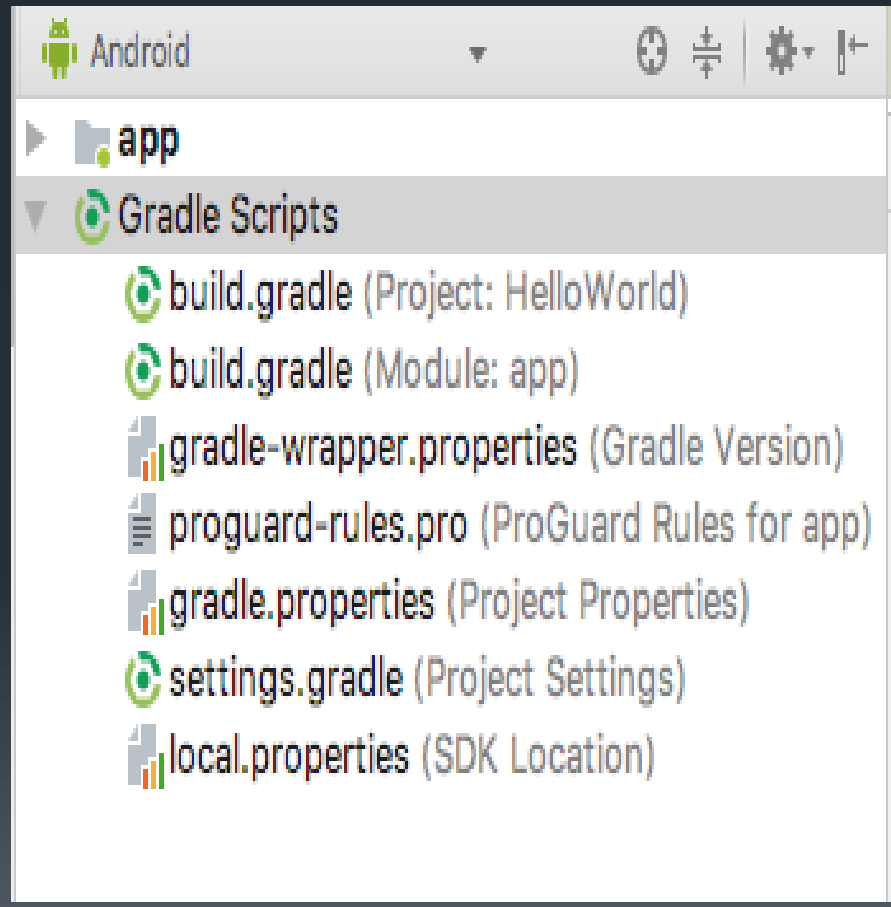
Explorer Views

- Explore the Project -> Android pane
- If not already selected, click the Project tab in the vertical tab column on the left side of the Android Studio window. The Project pane appears.
- To view the project in the standard Android project hierarchy, choose Android from the popup menu at the top of the Project pane, as shown below. [Default View for the App Development]

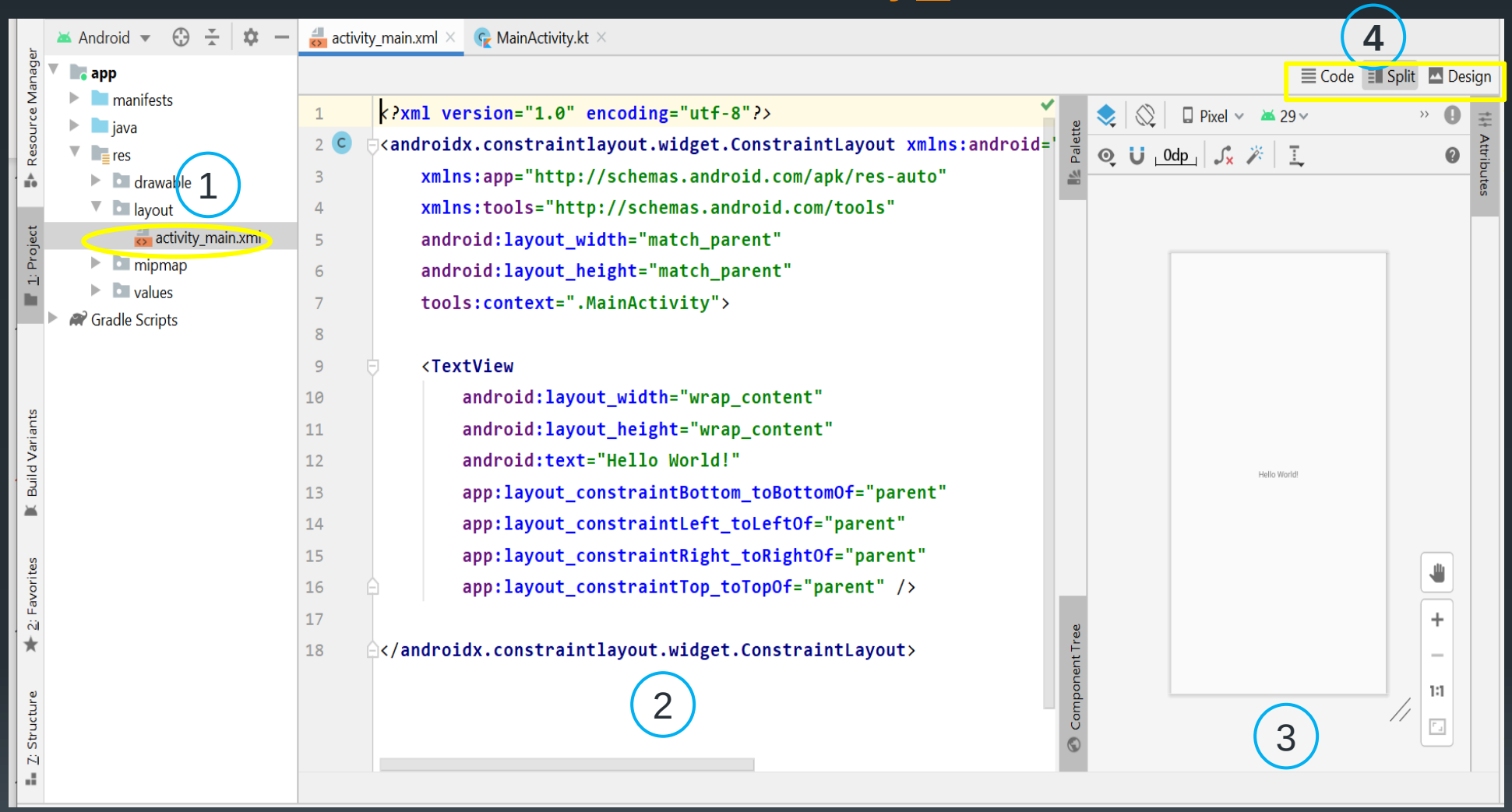


Gradle build system

- The Gradle build system in Android Studio makes it easy to include external library modules to your build as dependencies.
- Three build.gradle:
  - project
  - module
  - settings
- Learn more about gradle at https://gradle.org/
- Every Android project needs a gradle for generating an apk from the *.kt* and *.xml* files in the project. Simply put, a gradle takes all the source files (kt and XML) and apply appropriate tools, e.g., converts the java files into .dex files and compresses all of them into a single file known as apk that is actually used.

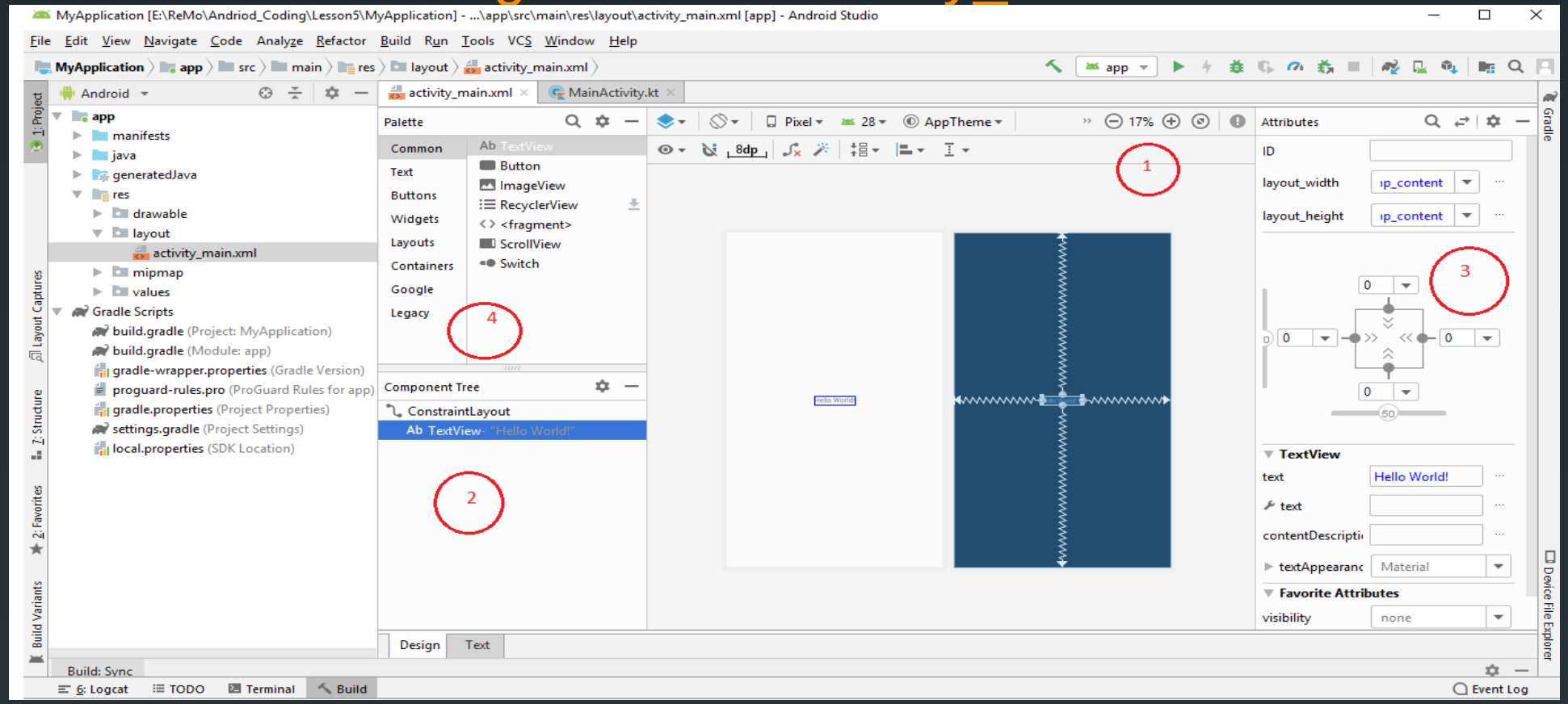


Code Editor - activity_main.xml



1. Click actitvity_main.xml to view the code editor and preview of Activity
2. XML editing panel -Type xml code to design activity
3. Preview panel and shows the current visual state of the layout
4. By clicking Design and Text button to switch between Design Editor and Code Editor. Split shows both Text and Design

Design Editor - activity_main.xml



1. **+** and **-** buttons for zooming in and out, or click the **Zoom to Fit Screen** button (right of the zoom buttons) so that both panels fit reasonably on your screen.
2. Component Tree - This panel shows the hierarchy of views in your layout.
3. Properties Panel - This panel displays the attributes assigned to the currently selected component in the layout.
4. The Palette panel – It consists of two columns with the left-hand column containing a list of view component categories. The right-hand column lists the components contained within the currently selected category.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">
    <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello World!"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent"/>
</android.support.constraint.ConstraintLayout>
```

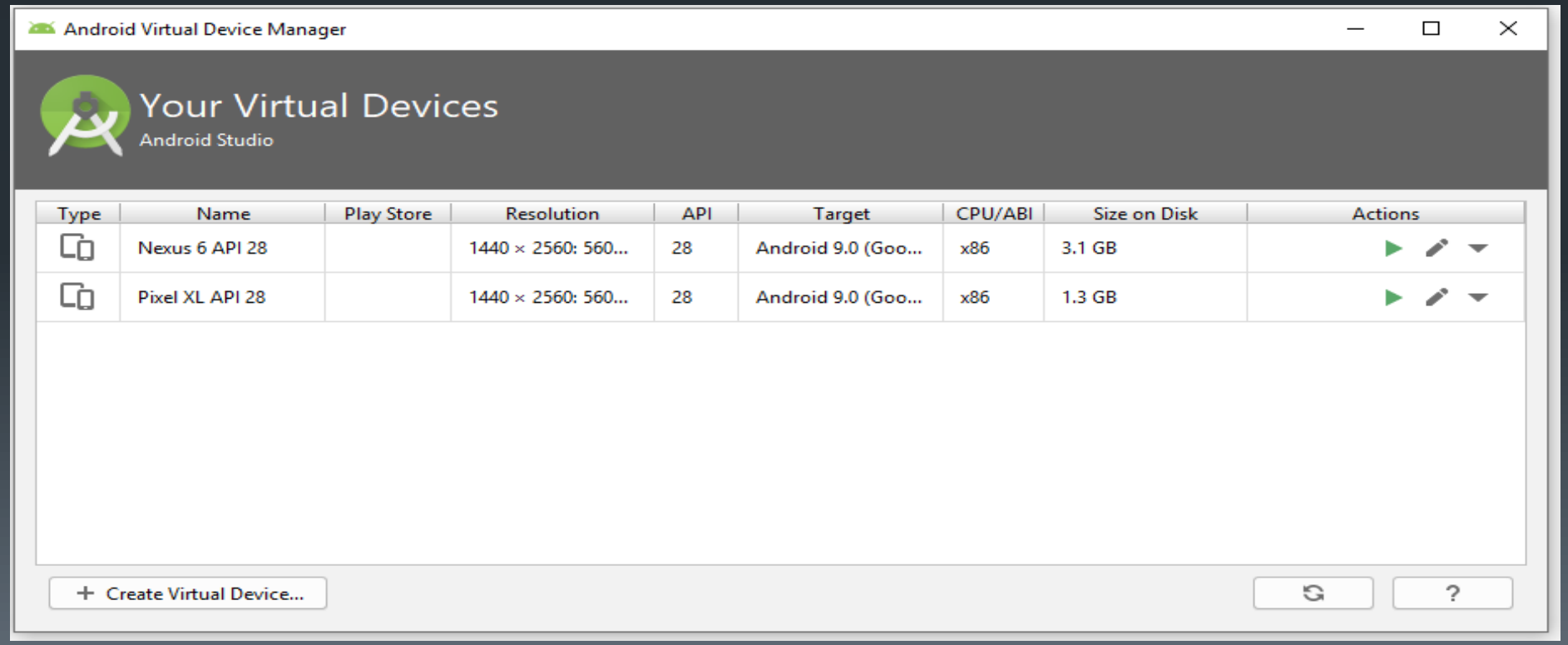
In the XML code, notice that the root element is `<android.support.constraint.ConstraintLayout>`.
The root element contains a single `<TextView>` element.
We will discuss about Layout and changing properties in the next lesson.

MainActivity.kt

```
package com.example.myapplication // Package name
// Need to import necessary libraries
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
// Every Kotlin Activity extends from AppCompatActivity class
class MainActivity : AppCompatActivity() {
// Must Override onCreate
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

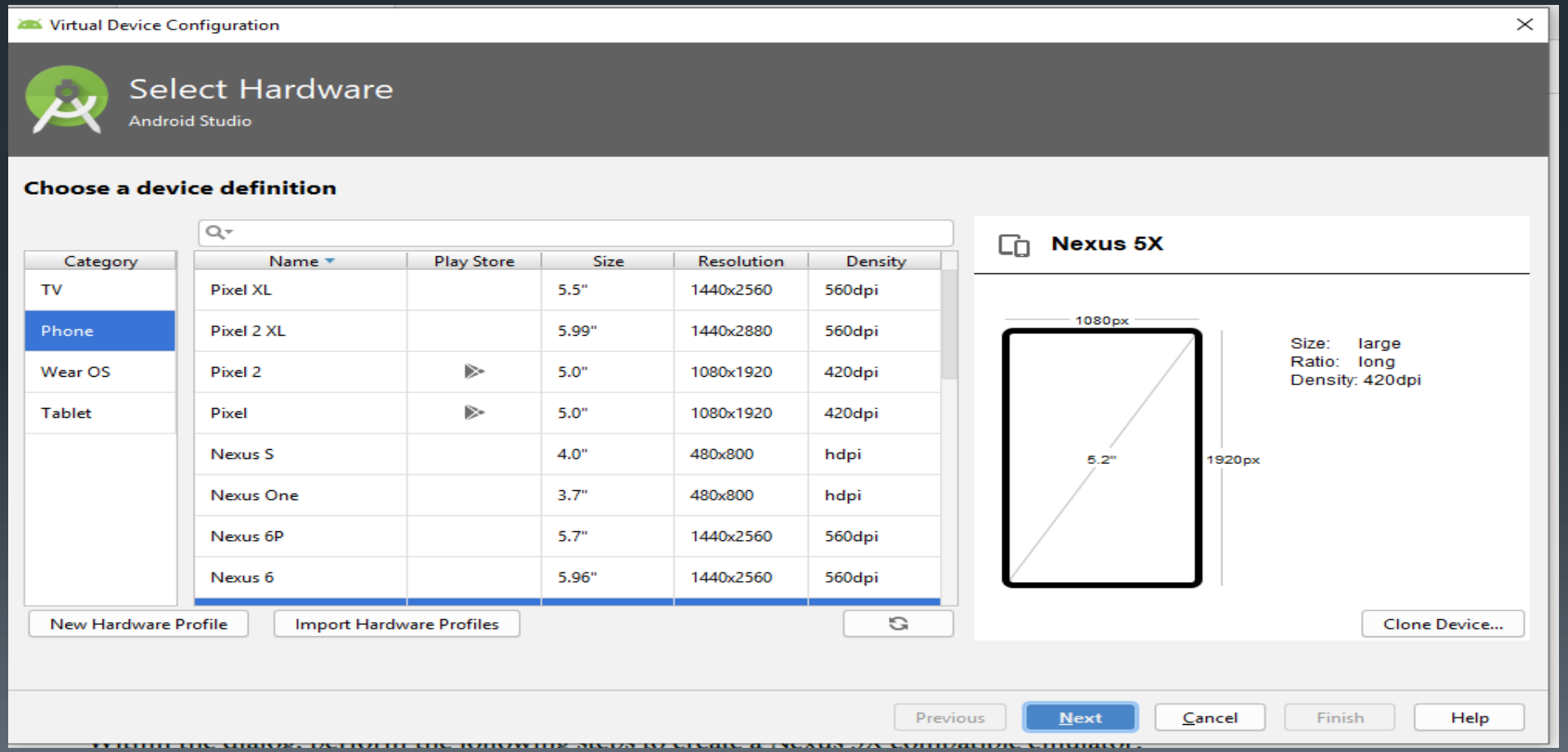
Open the Android Virtual Device Manager

- The AVD Manager allows you to set up new AVDs, and view and edit ones you've already created.
- Open it by selecting **Tools menu and choosing Device Manager/AVD Manager.**
- To add an additional AVD, begin by clicking on the *Create Virtual Device* button in order to invoke the *Virtual Device Configuration* dialog as shown in next slide



Open the Android Virtual Device Manager

1. Select the Device for Running App and click Next.
2. On the System Image screen, select the latest version of Android.
3. Click *Next* to proceed and enter a descriptive name (for example *Nexus 5X API 26*) into the name field or simply accept the default name.
4. Click *Finish* to create the AVD.



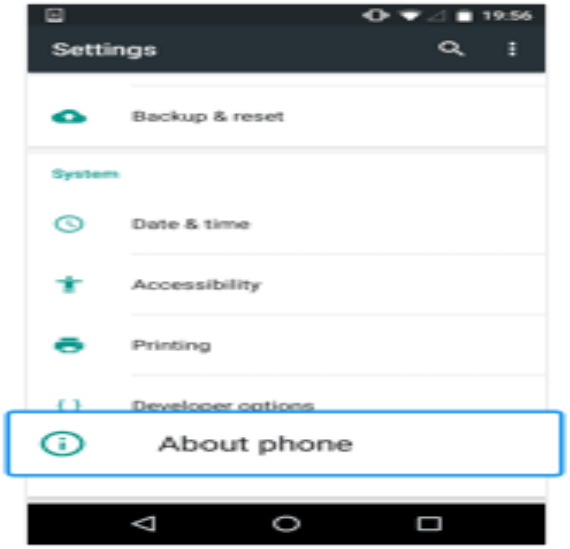
Running your App

- You can now run the app on a real device or on an emulator.

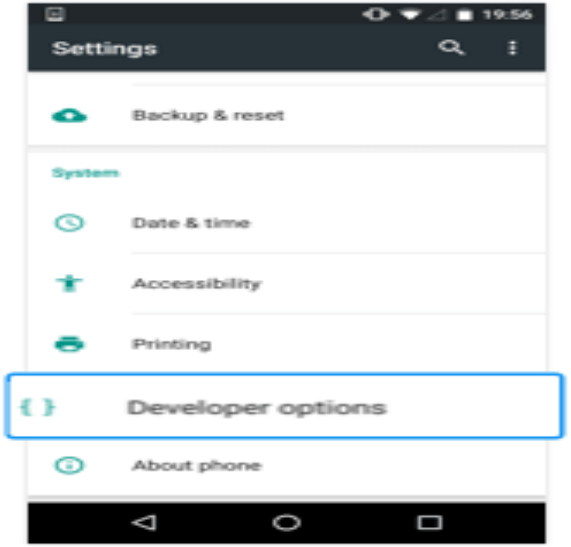
Run on a Real Device, Set up your device as follows:

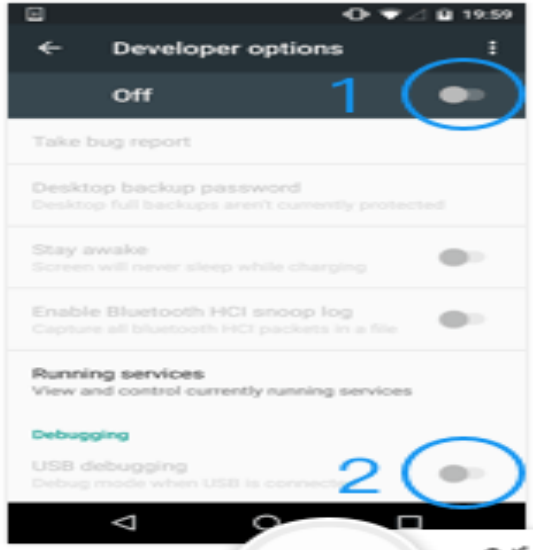
- Connect your device to your development machine with a USB cable. If you're developing on Windows, you might need to install the appropriate USB driver for your device. For help installing drivers, see the https://developer.android.com/training/basics/firstapp/running-app.html#RealDevice
- Enable **USB debugging** on your device by going to **Settings > Developer options**. **Note:** On Android 4.2 and newer, **Developer options** is hidden by default. To make it available, go to **Settings > About phone/Tablet/Device** and tap **Build number** seven times. Return to the previous screen to find **Developer options**.
- Enable **USB Debugging**.
- See the next slide for Pictorial representation to enable developer options

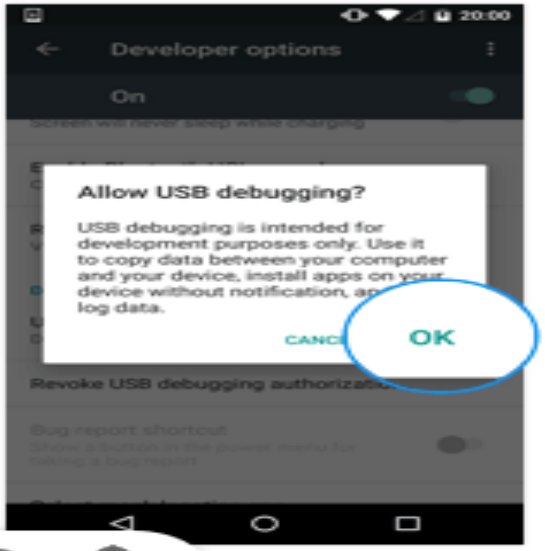
Running your App on Real Device Setup Screenshots



2. Go to "Developer options", turn it on, and then enable "USB debugging".







Running your App

Run the app from Android Studio as follows:

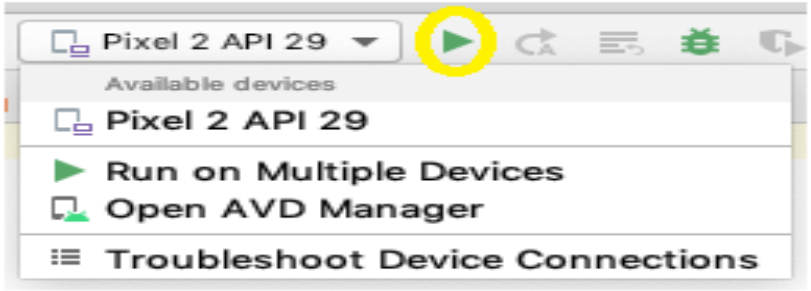
- In Android Studio, select your project and click **Run** from the toolbar.
- In the **Select Deployment Target** window, select your device, and click **OK**.
- Android Studio installs the app on your connected device and starts it.

Running and Deploy your App

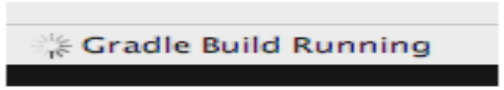
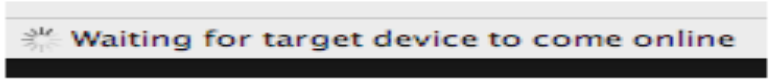
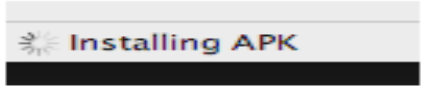
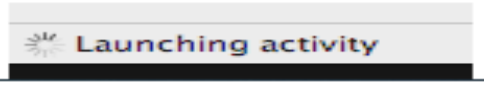
- Choosing the Run option doesn’t just run your app. It also deals with all the preliminary tasks that are needed for the app to run:
- ***Once you run your app a***n Android Application Package, or APK file, gets created. (Which is the installation file in Android for the OS)
- The file includes the compiled Kotlin files, along with any libraries and resources needed by your app.
- Once the emulator has been launched and the AVD is active, the APK file is uploaded to the AVD and installed.
- The AVD starts the main activity associated with the app.

Running and Deploy App

In **Run > Select Device**, under **Available devices**, select the virtual device that you just configured. This menu also appears in the toolbar.



The emulator starts and boots just like a physical device. Depending on the speed of your computer, this may take a while. You can look in the small horizontal status bar at the very bottom of Android Studio for messages to see the progress.

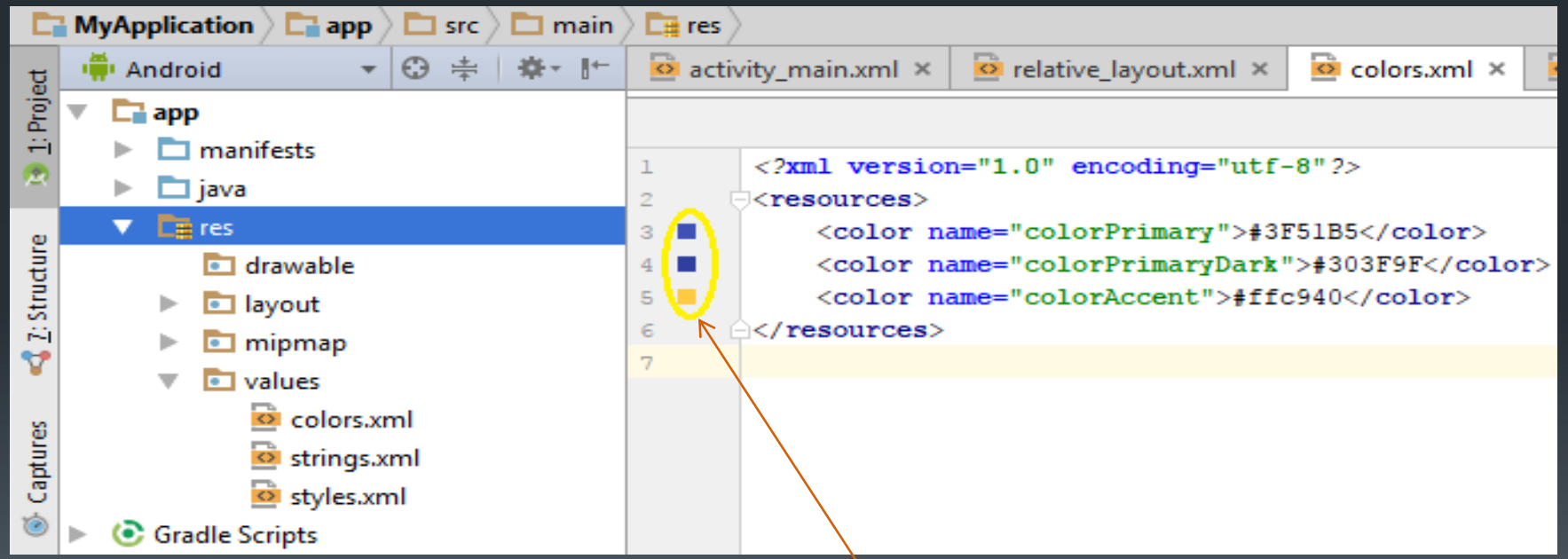
Messages that might appear briefly in the status bar	
Gradle build running	
Waiting for target device to come on line	
Installing APK	
Launching activity	

Main Point 1

- Learning Android programming is further growth in your programming knowledge to get a new experience of mobile programming. ***Science of Consciousness:*** *Similarly, experience is the practical basis of knowledge. With growth of experience, the abstraction of TC becomes more concrete.*

colors.xml

- Android uses standard RGB (red, green and blue) color model. Each primary color value is usually represented by hexadecimal number.
- At the beginning of such a color definition you must put a pound character (#).
- Color used in the layout xml file is maintained in the color.xml
- Example



To change the assigned color click the Squared colors

Hands on Example - BirthdayWishApp

- BirthDayWishApp contains
  - LinearLayout – To organize the components
  - EditText – To enter to whom you want wish
  - TextView – To show the Birthday wish Message
  - ImageView – To show the Giftbox
  - Button – Once the user click the SURPRISE button, you will get the Birthday Message with the given friend name appeared on the TextView and the giftbox will show the gift.
  - Space – To provide space between components
- This app will explain how to handle the click events.
- **Refer: UpdatedBWishApp**

Event Handlers

- Methods that do something in response to a click
  - A method, called an event handler, is triggered by a specific event and does something in response to the event
- Handle the Click event in two ways
  - In XML way
  - In kotlin way
- XML Way
  - Configure the following attribute to the UI component

    android:onClick="method_name"

    android:id="@+id/idname"
  - If you click the button, it will invoke the specified method from Kotlin code. You have to specify your action in code by creating inside MainActiviity.kt

    fun method_name(view : View) {//Implementation}
  - If the method is not available, it will throw java.lang.IllegalStateException: Could not find method
  - View is the parent class for all UI Group and components, by using this object you can call objects from xml.

activity_main.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"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:layout_margin="8dp"
        android:padding="20dp"
        tools:context=".MainActivity">
    <EditText
            android:id="@+id/name"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textSize="20sp"
            android:hint="Enter Your Friend Name"/>

    <Space android:layout_width="match_parent"
           android:layout_height="30dp"/>
```

activity_main.xml

```
<TextView
      android:id="@+id/msg"
       android:textSize="25sp"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"/>

<Space android:layout_width="match_parent"
      android:layout_height="30dp"/>

<ImageView
      android:id="@+id/image"
       android:layout_width="300dp"
          android:layout_height="300dp"
          android:src="@drawable/giftbox"
          android:layout_gravity="center" />

<Space android:layout_width="match_parent"
      android:layout_height="30dp"/>
```

activity_main.xml

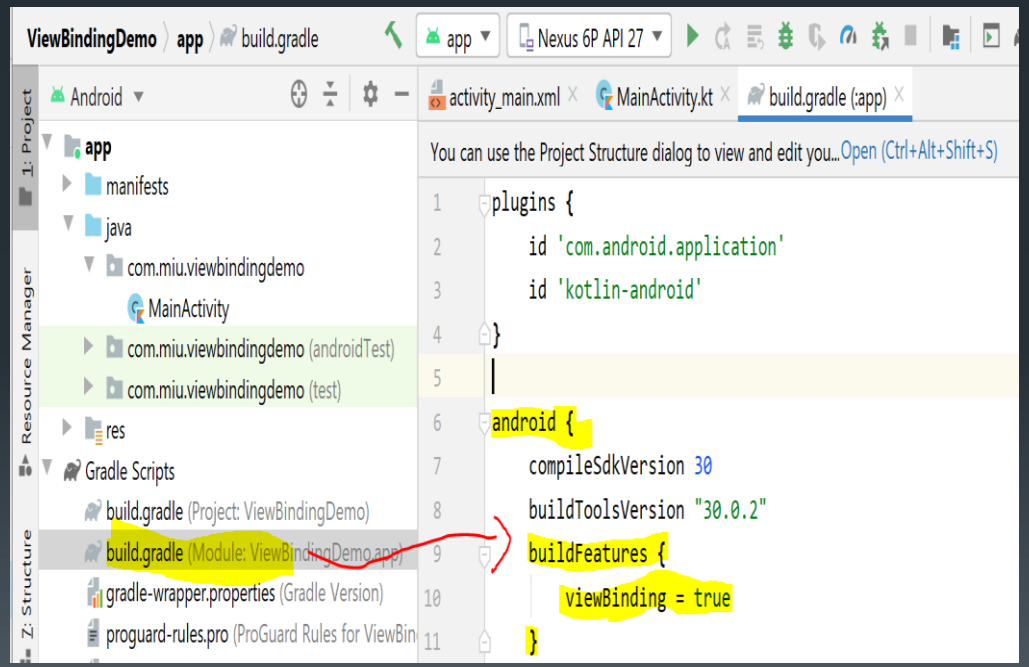
```
<Space android:layout_width="match_parent"
       android:layout_height="30dp"/>

<Button
       android:id="@+id/button"
       android:text="Surprise!"
       android:textSize="20sp"
       android:onClick="click"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"/>
</LinearLayout>
```

Note : The highlighted id can refer in Kotlin code

View Binding to access xml ids inside Activity

You need to enable the ViewBinding is true in build.gradle module as per the screenshot below. Android creates a Binding Class for your Layout. Bind the View in your Activity.

buildFeatures{ viewBinding=true } After adding the highlighted code, Sync your project.




```
// activity_main.xml
<ConstraintLayout>
   <TextView android:id="@+id/greeting">
</ConstraintLayout>


// generates AndroidMainBinding
class ActivityMainBinding {
    val greeting: TextView
}
```

```
override fun onCreate(savedInstanceState: Bundle?) {
    ...
    val binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)
```

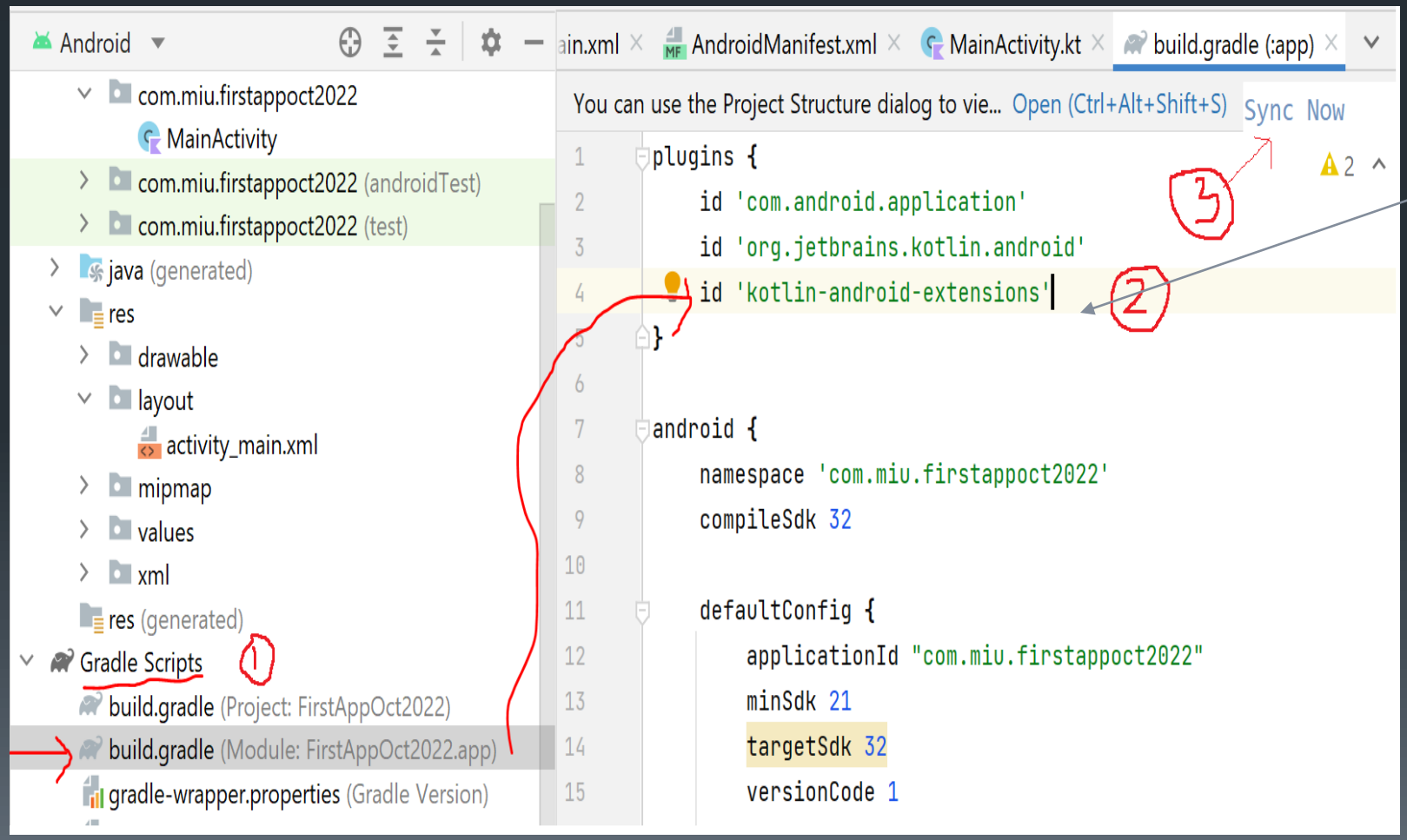
Always use the highlighted code in all Projects

Code to apply ViewBinding inside your Activity

```
class MainActivity : AppCompatActivity() {
    /* Get the instance of Binding Object,
    ActivityMainBinding is autogenerated ViewBinding class for the activity_main.xml*/
    private lateinit var binding: ActivityMainBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // Initialize the binding object with the given below code
        binding = ActivityMainBinding.inflate(layoutInflater)
        // Replace this line as mentioned below --> setContentView(R.layout.activity_main)
        setContentView(binding.root) // root is your root layout specified in activity_main.xml
         binding.tv.text = "Sample View Binding Demo" // tv is the id from XML
    }
}
```

Old Approach - To get xml id access in Kotlin file

Note : The highlighted id can refer directly in Kotlin code by importing **import kotlinx.android.synthetic.main.activity_main.*** in your activity by adding the highlighted apply plugin lines in the build.grade(Module) as per the below screenshots and **click Sync Now** on the top.



id 'kotlin-android-extensions'

Handling Event XML Way - MainActivity.kt

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
}
      /* Button click event implementation - XML Way
      android:onClick="click“. It is the function name should use in Kotlin
      code  to perform event click*/

    fun click(view : View){
            // Change the Label Text
         msg.text = "Happy BirthDay Dear ${name.text}!"
            // Change the ImageView dynamically
        image.setImageResource(R.drawable.gift)
    }
}
```

Running the App

Before Clicking
SURPRICE Button.
Type your Friend Name

After Clicking
SURPRICE Button.

Old approach - findViewById

findViewById(int id) is very useful function to access or update properties of Views. Views are LinearLayout, TextView, Button, EditText, ImageView etc. To find a view programmatically, should have been set with an id in the layout xml file

```
<ImageView
        android:id="@+id/image"
        android:layout_width="300dp"
          android:layout_height="300dp"
          android:src="@drawable/giftbox"
          android:layout_gravity="center" />
```

Get the reference to the view and use that reference to access its properties

Inside your Kotlin code

```
var image =  findViewById<ImageView>(R.id.image)

    image.setImageResource(R.drawable.gift)
```

In Kotlin you may skip and not use the **findViewById( )** and still be able to refer to UI components defined in XML Layout file by Import the data binding library into the Kotlin file (import kotlinx.android.synthetic.main.activity_main.* in your activity)

Handling Event Kotlin Way - MainActivity.kt

```
class MainActivity : AppCompatActivity() {
private lateinit var binding: ActivityMainBinding
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)
    binding.button.setOnClickListener {
        // Change the Label Text
        binding.msg.text = "Happy BirthDay Dear ${binding.name.text}!"
        // Change the ImageView dynamically
        binding.image.setImageResource(R.drawable.gift)
    }
  }
}
```

Summary Points

- A typical Android app is comprised of activities, layouts, and resource files.
- Layouts describe what your app looks like. They're held in the app/src/main/res/layout folder.
- Activities describe what your app does, and how it interacts with the user. The activities you write are held in the app/src/main/java folder.
- AndroidManifest.xml contains information about the app itself. It lives in the app/src/main folder.
- An AVD is an Android Virtual Device. It runs in the Android emulator and mimics a physical Android device.

Summary Points

- An APK is an Android application package. It’s like a JAR file for Android apps, and contains your app bytecode, libraries, and resources. You install an app on a device by installing the APK.
- Android apps run in separate processes using the Android runtime (ART).
- The TextView element is used for displaying text.
- EditText, a user interface element for entering and modifying text
- LinearLayout aligns all children in a single direction, vertically or horizontally. You can specify the layout direction with the android:orientation attribute.

Main Point 2

- XML in Android allows developers to create user interface that are rich in content and functionality with the help of styles, colors and event handling. *The ultimate provider of tools for the creation of beautiful and functional content in manifest existence is pure intelligence itself; all creativity arises from this field's self-interacting dynamics.*

UNITY CHART

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

Synthesis of parts for completeness of living

1. The Manifest, Activity, Views, and Layouts are distinct parts of an Android program just as Creative Intelligence synthesizes parts for completeness of living.
2. The many parts of an Android program function together in a cohesive whole as an Android app just as Creative Intelligence binds together delicate impulses of life.

---

3. **Transcendental Consciousness** *is the field of pure intelligence, the source of all activity.*
4. ***Impulses within the Transcendental field*** *can be viewed as the operations of the cosmic computer*
5. ***Wholeness moving within itself****: In Unity consciousness we perceive the world as an expression of our own infinite Self.*