Lesson 5: Creating User Interface

Introduction

Lesson 5 focuses on how to build a user interface using Android layouts for all types of devices. Android provides a flexible framework for user interface design which allows your application to display different layouts for different devices, create custom user interface widgets, and even control aspects of the system UI outside your app's window.

All user interface elements in an Android application are built using View and ViewGroup objects. A View is an object that draws something on the screen that users can interact with. Android provides a collection of classes and subclasses which offers common input controls (such as buttons and text fields) and various layout models (such as a linear or relative layout).

Android Project Structure

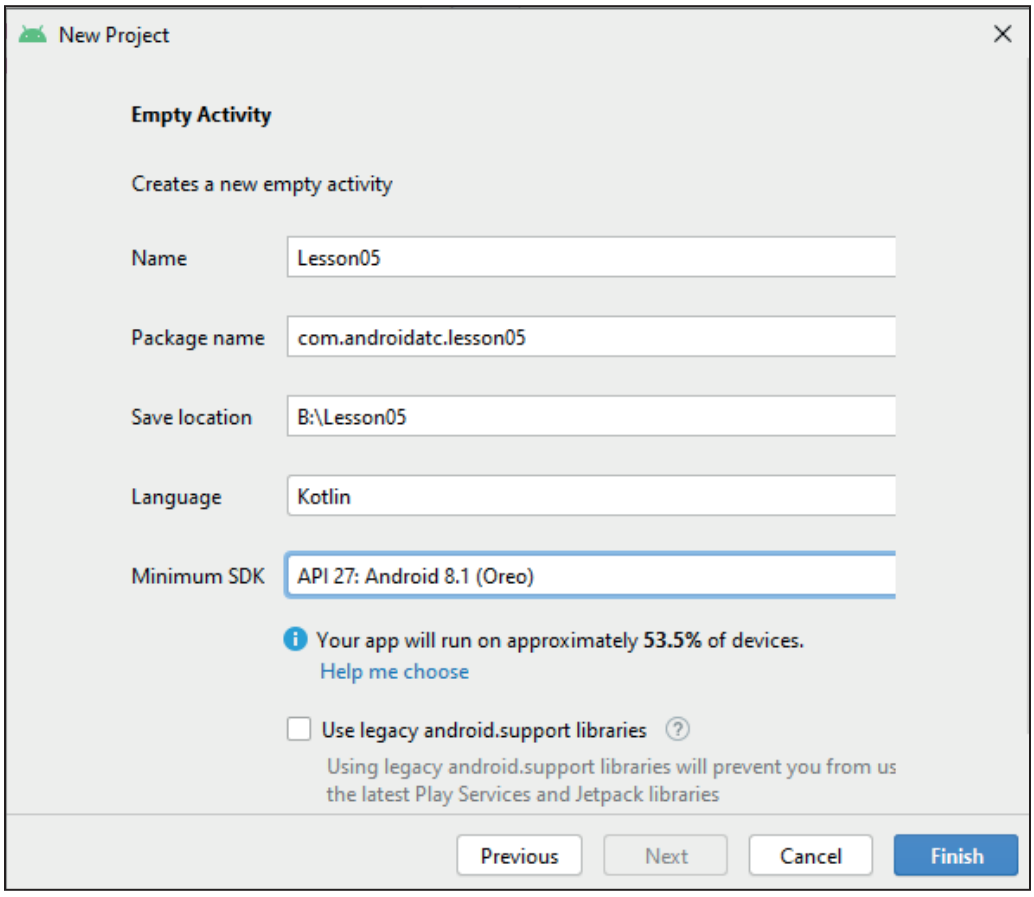
Android project structure is similar to the website design structure. When you design a website, first you have to implement a plan for this website structure or content by creating different folders for different types of files. For example, you have to create a folder for the HTML pages, a folder for images, another folder for animation, and possibly another one for the database.
Using Android Studio, we do not need to worry about that. When you create a new Android project using Android Studio, a set of folders and files are created automatically. The created files will include everything that defines your Android application, from application source code to build configurations and application resources.

In the previous lesson, you created an Android app for a simple calculator. In this lesson, you will create a simple Android application, and you will have to go deeply in the structure of your project files and folders. This will give you an idea about the importance of each file or folder and how you may use any of them to edit or configure your app as well.

The following steps show how you can create a new Android application:

1- Open Android Studio, and then select **File → New → New Project**
2- Select **Empty Activity**, and click **Next**
3- In the following wizard, type the application name **Lesson02**, company name, and select the project location as illustrated in the following figure, then click **Finish**.

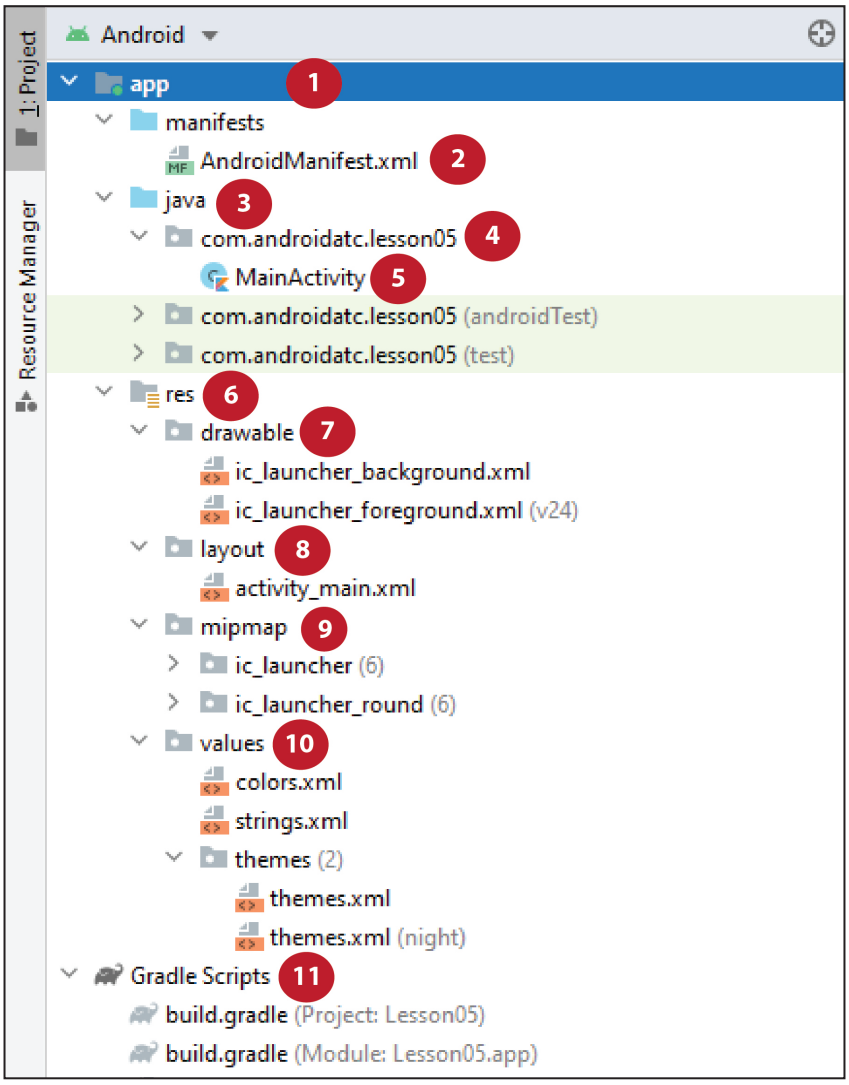
Note: In this step, keep the default configuration, as illustrated in the figure below. You are creating an Android application that can work on any phone or tablet with the lowest API level 27 or has Android OS version 8.1.



Note: You can change the name of your app later after your application is created by changing the `android:label` attribute value in the AndroidManifest.xml file (app → Manifests → AndroidManifest.xml)

A project in Android Studio contains everything that defines your workspace for an Android app from source code and assets to testing code and building configuration. When you start a new Android project, Android Studio creates the necessary structure for all your files and makes them visible in the Project window on the left side of the IDE (click **View → Tool Windows → Project**). This page provides an overview of the key components inside your project.

The following figure shows the Android project structure content:



The following are the components of Android project structure with a brief description of each:

1- Android **app** module: It provides a container for your app's source code, resource files, and app level settings such as the module-level build file and Android Manifest file. When you create a new Android project, the default module name is "**app**".

2- **AndroidManifest.xml** : Every application must have an AndroidManifest.xml file (precisely with this name) in its root directory. The manifest file provides essential information about your application to the Android system. The system must read this information before it can run any of the application's code. This XML file includes the Java package names of the application which are unique identifiers for the application, a detailed description of the application's components (activities, services, broadcast receivers, and content providers), and the permissions needed for the application to access protected parts of the Android system. We will discuss the content and settings of this file later in details in the next lessons of this course.

The following is the **Android Manifest.xml** for Lesson05_app:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidatc.lesson05">
<application
   android:allowBackup="true"
   android:icon="@mipmap/ic_launcher"
   android:label="@string/app_name"
   android:roundIcon="@mipmap/ic_launcher_round"
   android:supportsRtl="true"
   android:theme="@style/Theme.Lesson05">
   <activity android:name=".MainActivity">
     <intent-filter>
         <action android:name="android.intent.action.MAIN" />

         <category android:name="android.intent.category.LAUNCHER" />
     </intent-filter>
   </activity>
</application>

</manifest>
```

3- **Java** Directory: It contains the application source code files, separated by package names.

4- **com.androidatc.lesson05**: This is the app package name. It is just a unique identifier for your application at Google Play Store. It includes the Kotlin source files for your app.

5- **MainActivity.kt** : This is the actual application source file which ultimately gets converted to a Dalvik executable and runs your application. It includes the code of the default startup app interface.

6- **res**: This directory includes other various XML files which contain a collection of resources, such as strings and colors definitions.

7- **Drawable**: This is a directory for drawable objects that are designed for high-resolution screens. This directory may contain bitmap files (png, gif,or jpeg) or XML files that are compiled into some drawable resource like re-sizable bitmaps, shapes or animation drawable files. You will use this file directory later in this lesson to save your app images.

8- **Layout**: It includes the XML files that define a user interface layout of activities such as **activity_main.xml** file which includes the user interface of the MainActivity file.

9- **Mipmap**: This folder includes drawable files for different launcher icon densities (your app icon). We will learn later in this lesson how to configure your app icon using this directory.

10- **Values**: This folder includes the main XML files that contain simple values such as strings, colors and styles. Later, you will work with these XML files to adjust the activity format.

11- **Gradle Scripts**: Android Studio uses Gradle, an advanced build toolkit, to automate and manage the build process while allowing you to define flexible custom build configurations. Each built configuration can define its own set of codes and resources, while reusing the parts

common to all versions of your app. Android plugin for Gradle works with the build toolkit to provide processes and configurable settings that are specific to building and testing Android applications.

View

All user interface elements in an Android application are built using View and ViewGroup objects. A View is a public class that draws something on the screen where the user can interact with. View is the base class for widgets, which are used to create interactive user interface components (buttons, text fields, etc.). All of the views in a window are arranged in a single tree. You can add views either from code or by specifying a tree of views in one or more XML layout files.

There are many specialized subclasses of views that act as controls or are capable of displaying text, images, or other contents. Once you have created a tree of views, there are typically a few types of common operations you may wish to perform using Kotlin methods or XML attributes.

Adding a View to your application

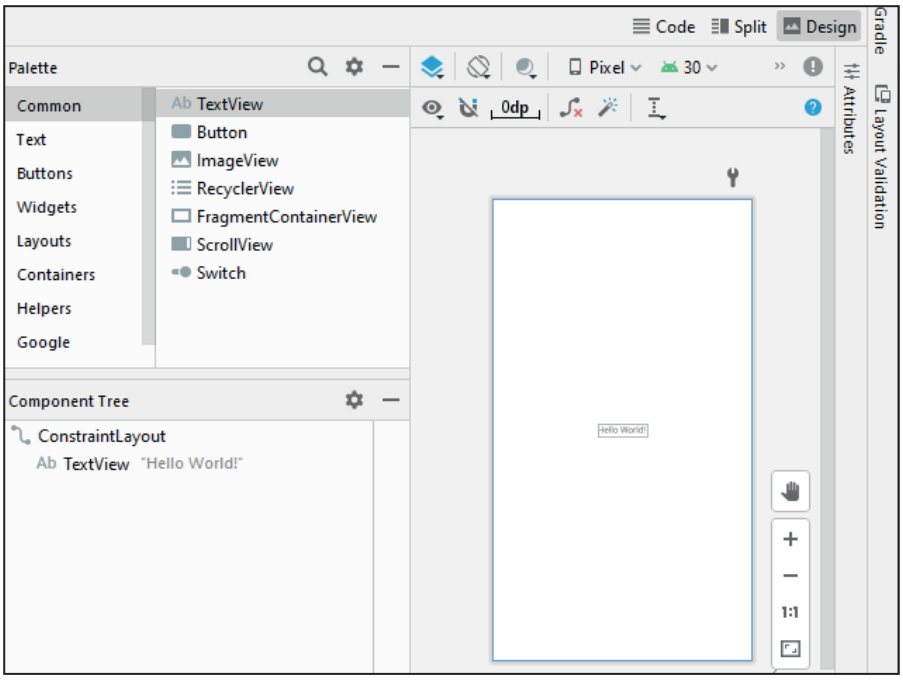
There are two ways to add a View (buttons, text fields, etc...) to your activity:

Adding a View in an XML layout file

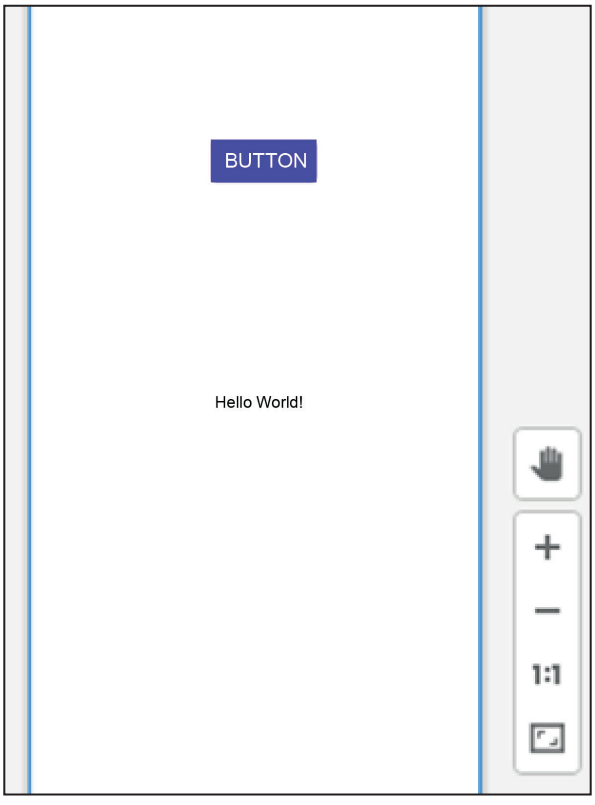
You have used this method previously to create a simple calculator in Lab 4. It is the easiest way for any Android developer to add a widget to an app because, simply, the developer will use dragging and dropping technique to add the ready widgets like buttons, checkbox and etc... to his/her app activity.

Example: Creating a simple hello button from the XML layout file:

1- Open the **activity_main.xml** file (app → res → layout → activity_main.xml) in **Design** view. You should have the following layout:



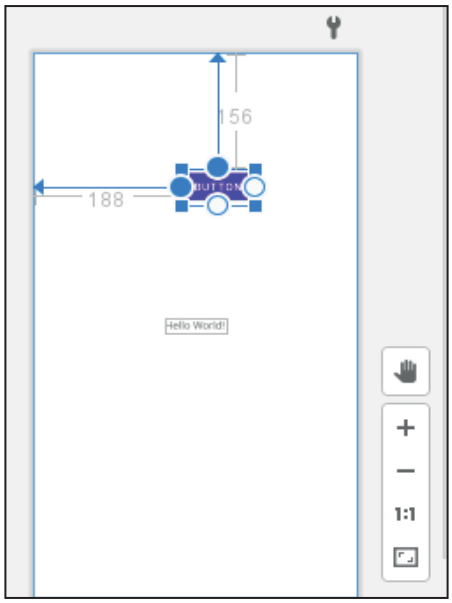
2- Drag a button from the **Palette** panel and drop it at the top of the activity layout. You should get a similar one to the following figure:



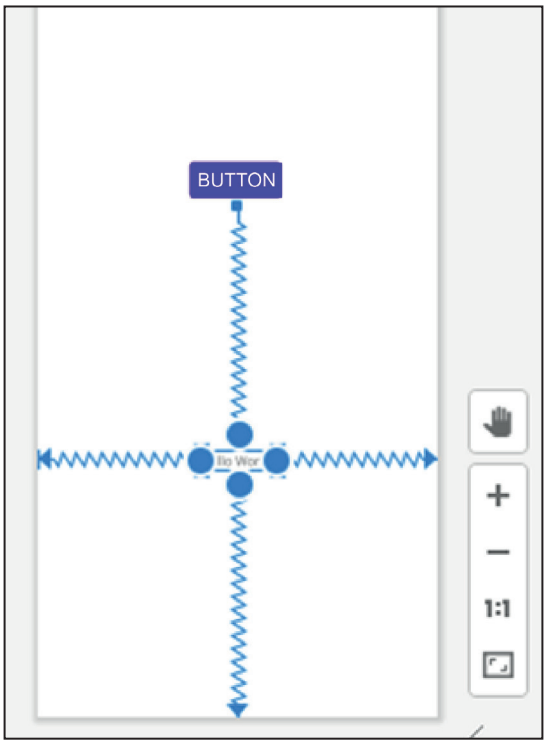
You find an error message in the Problems console displaying that this view (button) is not constrained. It only has design time positions, so it will jump to (0,0) at runtime unless you add the constraints.

This means you should add constraints (margins) for any view you may add to your activity. It is enough to set the top or down margin and set the right or left margin. These will determine exactly where this view (button) will be located when this app runs.

To do that, just drag the constraint arrow from this button border to the top or left edge of this activity. You should return this button to its previous location each time. You should get something similar to the following figure:



If you have another view such as another button or image above or below this button, you may set this button margin or constraint related to this view. For example, let us set the top constraint of the **TextView** (Hello World!) to be related to the button as illustrated in the figure below:



Adding a View using Kotlin code

This way is difficult and needs more development time because it requires to import some Kotlin classes and type many lines of code to create a simple view like button.
Can you imagine how would it be if you have to write a code every time you want to create a simple button? How much time will it take to create a complete application with many activities and widgets? It would be a nightmare.

Configuring Layout Views

The most important configuration for any view is the view ID because your layout may have a lot of buttons or other views; therefore, you should distinguish each view by known ID; otherwise the default generated ID to be used later when you start writing the Kotlin code, where you will call each of these views in your app code using its ID.

For example, open the XML layout file: **activity_main.xml** in the **Code** mode, and you should get the code below for the button. The gray highlighted code is for the button ID. Here its ID is: **button** and the button label is: **Button**.

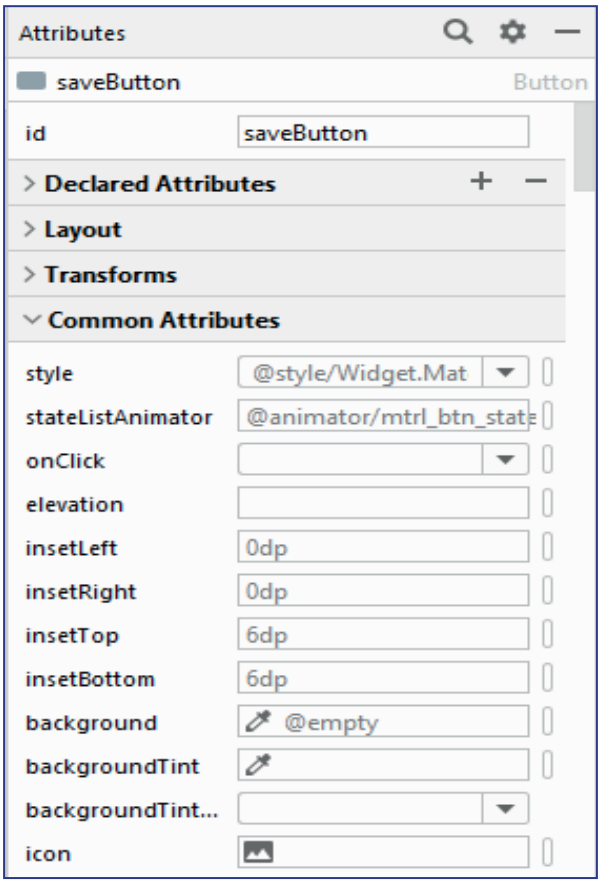
```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="156dp"
    android:layout_marginTop="156dp"
    android:text="Button"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

You may change the ID and the button label in previous XML code for this button as illustrated in the highlighted gray color in the following code:

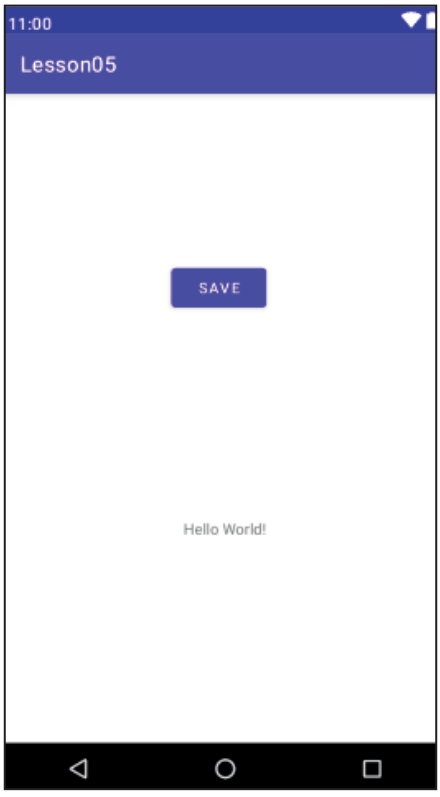
```
<Button
    android:id="@+id/saveButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="156dp"
    android:layout_marginTop="156dp"
    android:text="Save"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Note: Your view ID must be unique in your application.

Also, if you click the **Attributes** tab at the top right side (click the Design mode first), you can enter all the configurations for your view easily without needing to add a lot of XML attributes to get a specific format as illustrated in the figure below:



If you run your app, you should get the following figure:

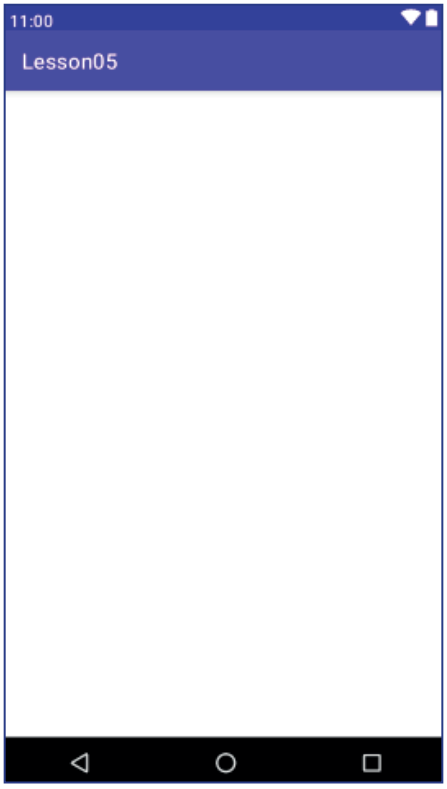


Creating a User Interface

In this step, you will use the Android Studio Layout editor to create the activity layout (XML file) which includes views such as text box, button, checkbox and image. Instead of writing XML codes to create each view, you will use Android Studio's Layout editor (Design mode) which makes building a layout easier by simply using the drag-and-drop views option.

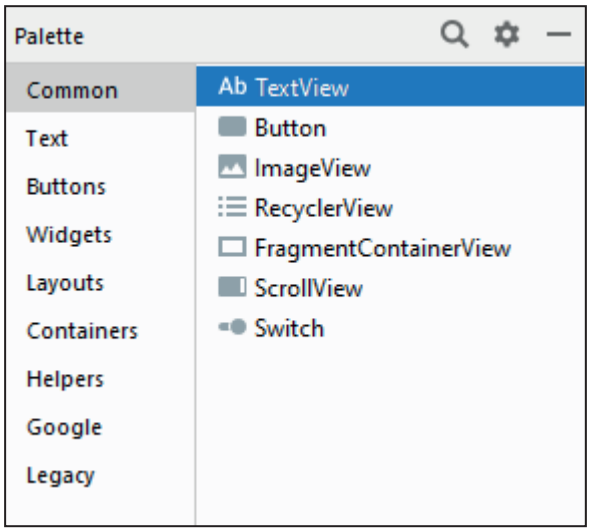
In addition, you will make the application respond or interact with these views when you write a suitable activity Kotlin code for each.

The following example includes steps on how to create a user interface with some widgets. Make sure to delete any previous buttons you have created before in **lesson05** and the run result will appear similar to the following figure:

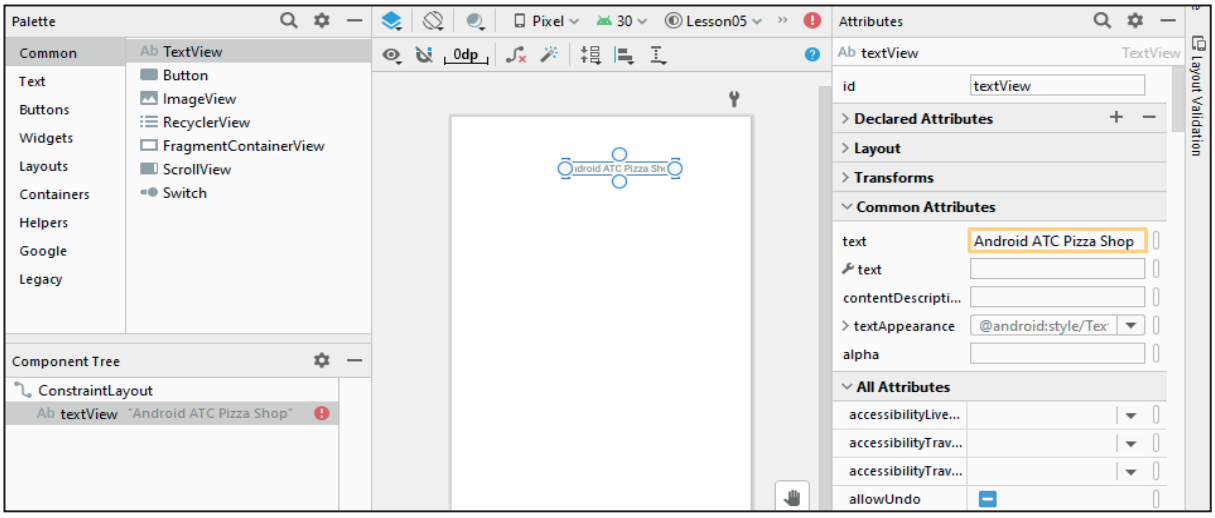


Adding a Text Box

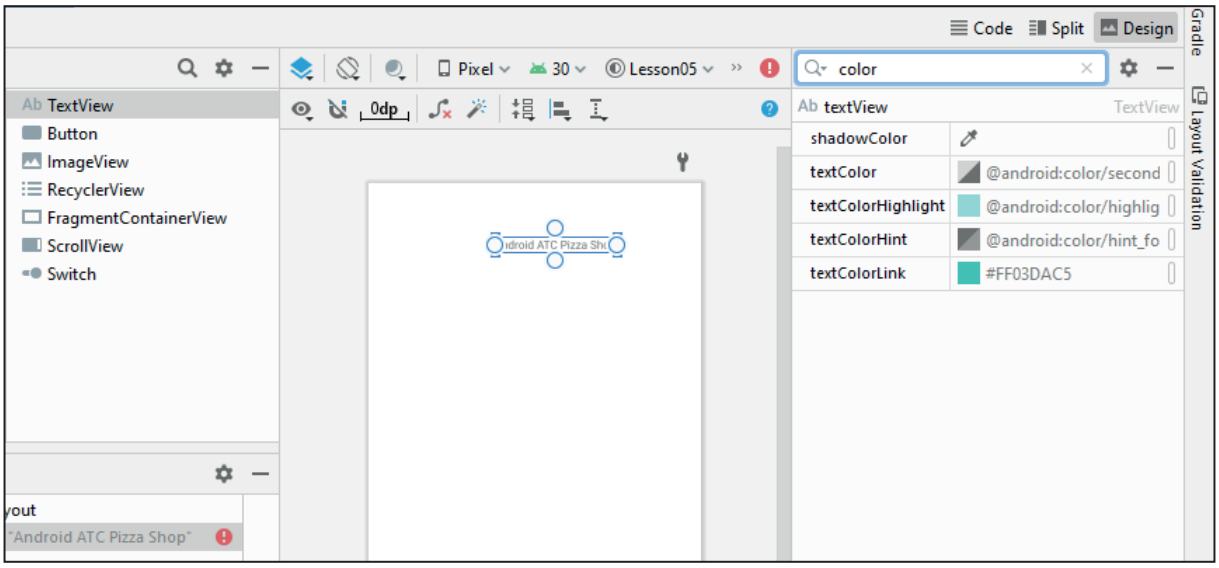
1-From the palette panel, and as illustrated below, select "**TextView**" then drag it to the activity layout.



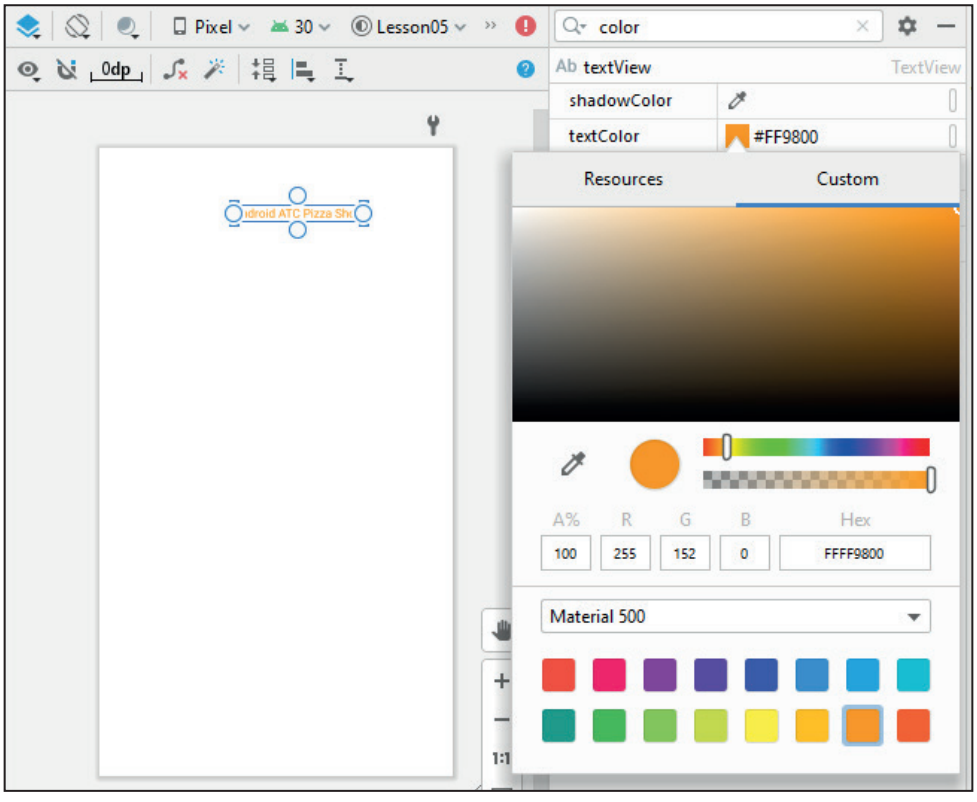
2- You can edit the attributes of this **TextView** (text box) using the **Attributes** panel located on the top right side of Android Studio as illustrated in the following figure:



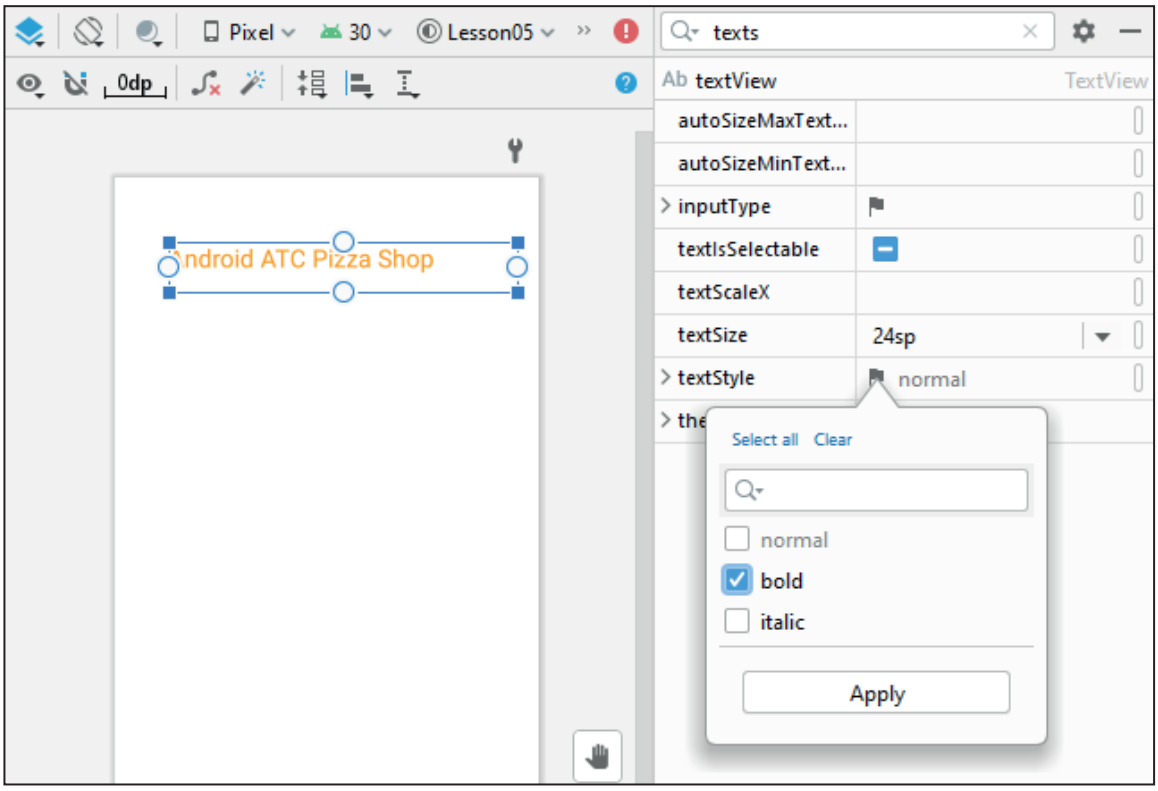
3- You may use the search area to find the attributes you want to modify like text color. Type **color** in the search area then press **Enter** (or return for Mac), and you will get "**textColor**" attribute as illustrated in the following figure:



4- Click "**Pick a Resource**" button to get the list of colors for the **textColor** attribute. Just select the text color you want as illustrated in the figure below:



You may use the same method to search for "**size**" to modify the "**textsize**" attribute, "**textStyle**" to change the font style to bold, italic or all caps, or "**textAlignment**" to align the text to center, left or right as illustrated in the figure below:

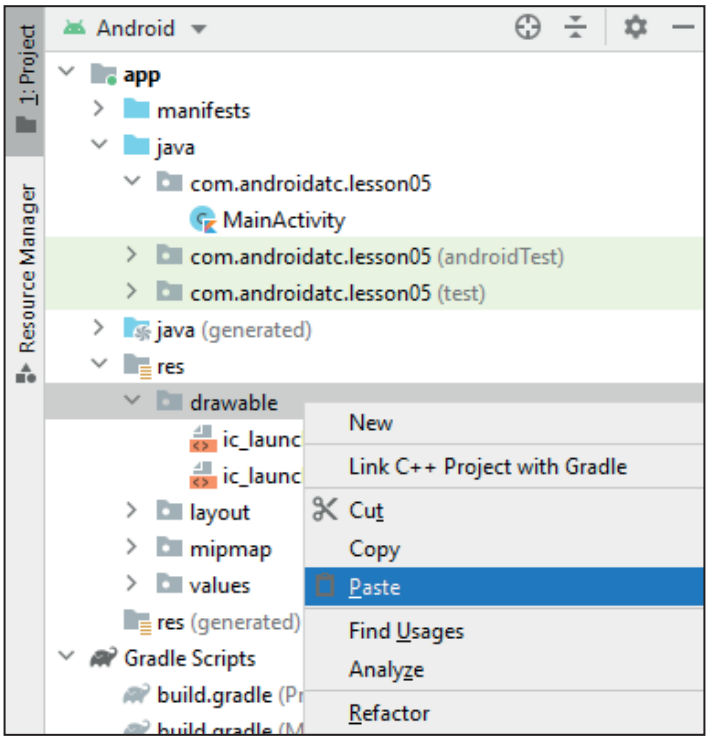


Adding an Image

Adding images to your app will add value and interactivity to your app users.
Can you imagine what the first impression for your app users may be if they run your app, and then run first interface of your app included texts only without any image? Adding images to your app interface is very important because it adds value to your app; in addition, if the image itself is selected carefully, it saves a lot of data or explanation in your app. Remember, the app users will not read a lot of data, they need to find or navigate your app to get your app service as soon as they can within the shortest time; therefore, make sure your app is easy to navigate with less text and with clear and valuable images.

The image which you will add to your app may be company logo, or icon or it may be used to add interactivity to your activity layout.

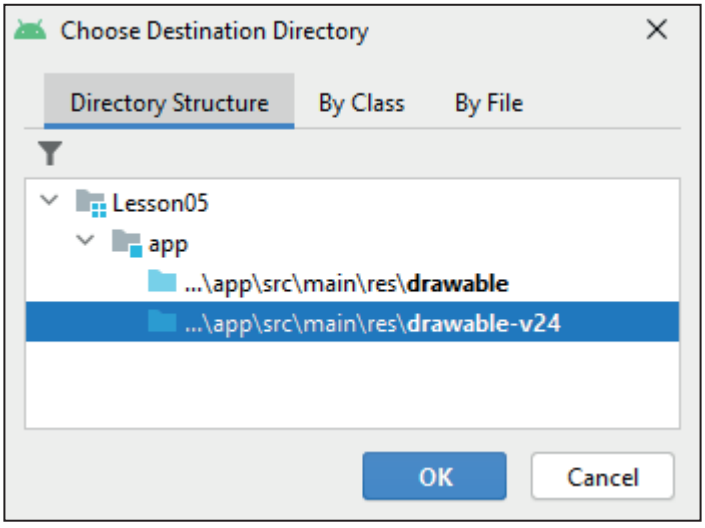
The image which you want to add to your app must be a part of your app files. You must add the image file first to your app files, and then add it to your app interface or activity.
You can add an image to your Android app by simply copying the image from the folder where it is stored in your computer, and then pasting it to the images location on the Android Studio: **app\res\drawable** folder as illustrated below:



To add an image to your app, follow these steps:
1- Browse your computer then select the image you want to add and copy it.

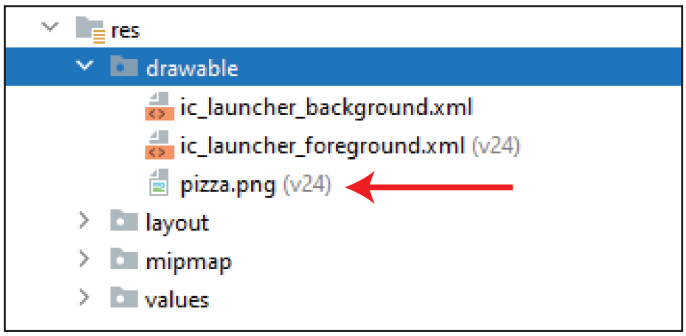
For example, right click "**pizza.png**" image file, then select **Copy**.

2- When you right click the **drawable** container (app → res → drawable) in Android Studio, and then click **Paste**, you will get the figure below. Select **drawable-24**, and then click **OK**.

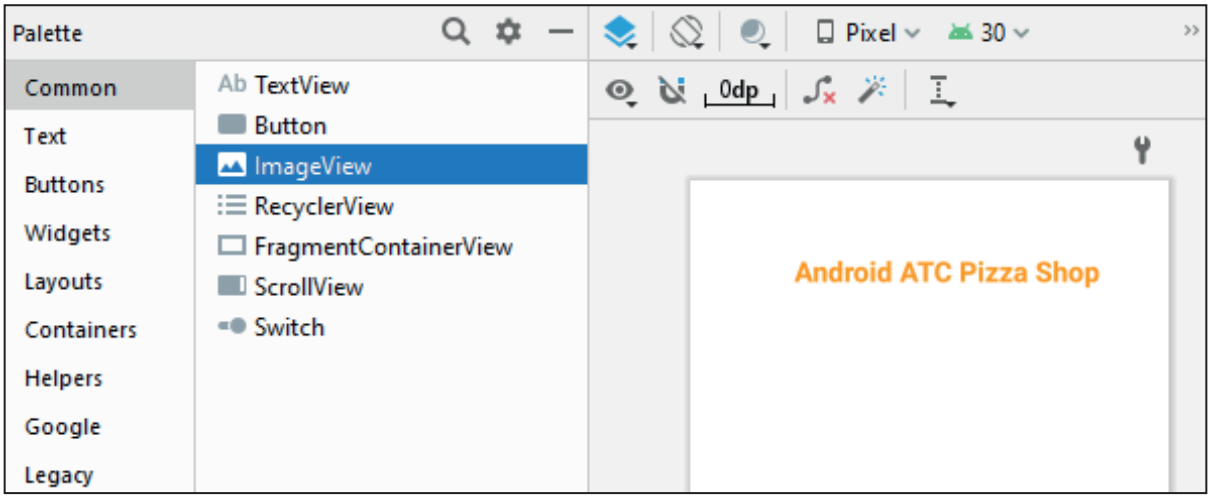


3- In the next step, click **OK**.

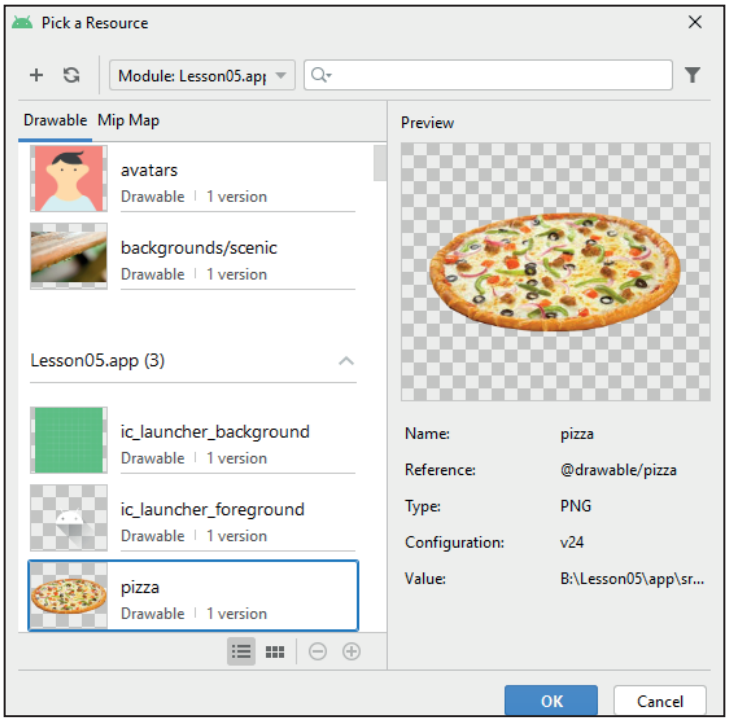
4- You get the following figure which displays the image you added to your application resources.



5- To insert this image into your app interface, drag and drop **ImageView** from the palette to your activity layout, as illustrated in the following figure:



6- You get the following "Pick a **Resource**" dialog box. Select the image (pizza image), then click **OK**.



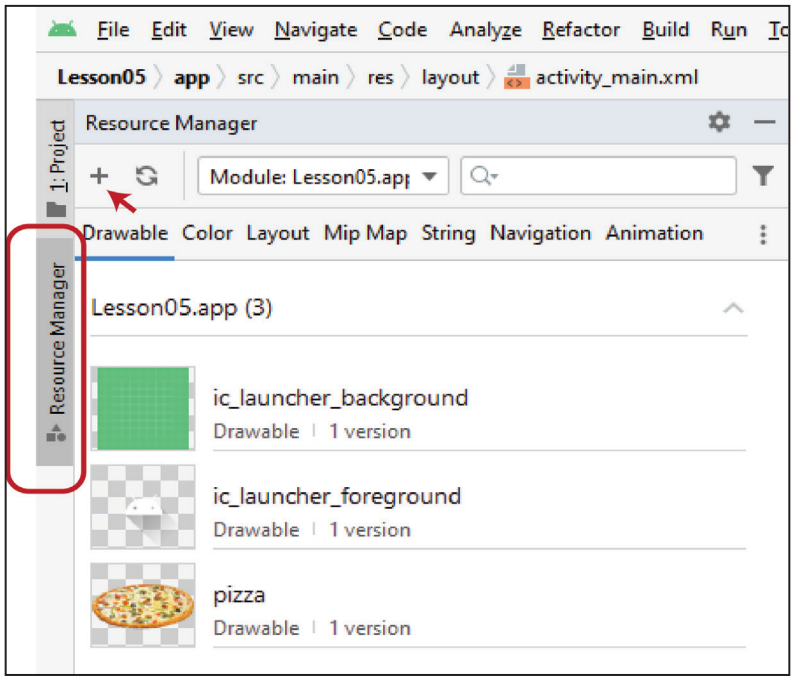
7- Your activity layout will look as the following figure:

9- Resize handles on each corner to set the margins (constraints) of the image, and then run your app. You will get the following figure:

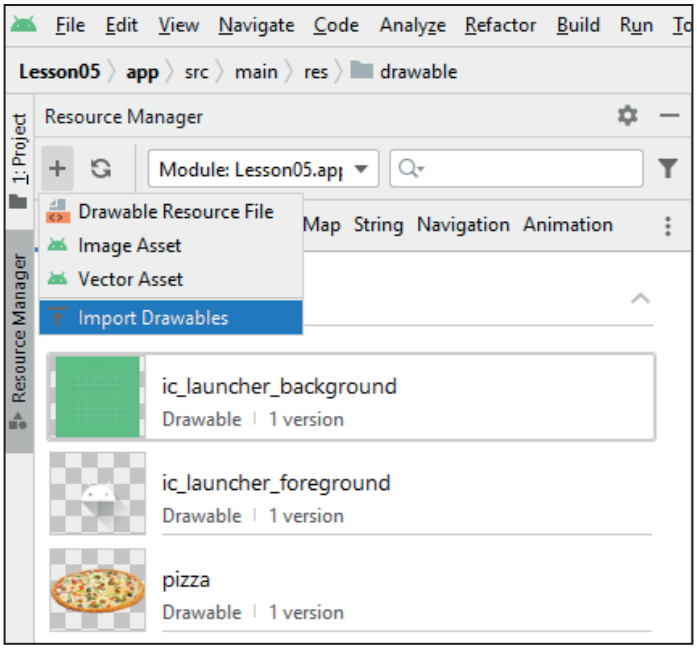
Manage your app's UI Resources with Resource Manager:

You may use the resource manager tool to import, manage, create, and use resources such as images, animation files, font files, colors, and other resources to your app user interface in a professional and easy way.

You can open the tool window by selecting **View → Tool Windows → Resource Manager** from the menu bar or by selecting **Resource Manager** on the left side bar as illustrated in the following figure:

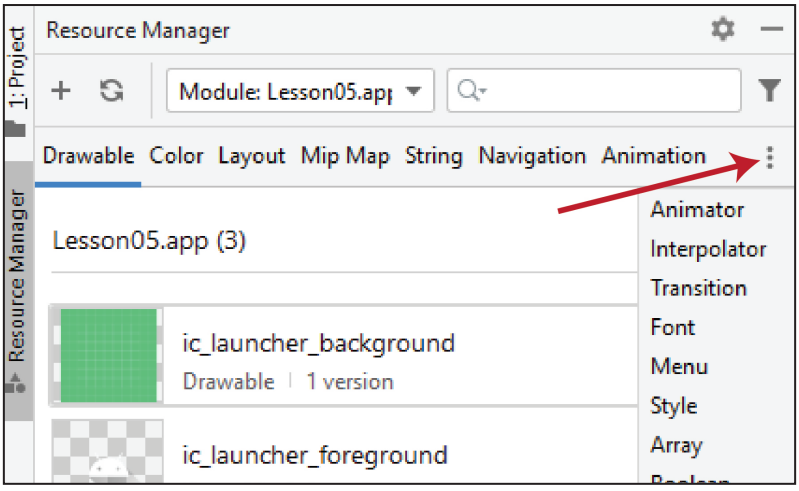


If you select the **Drawable** tab, click the plus icon (**+**), then select **Import Drawables** as illustrated in the figure below:



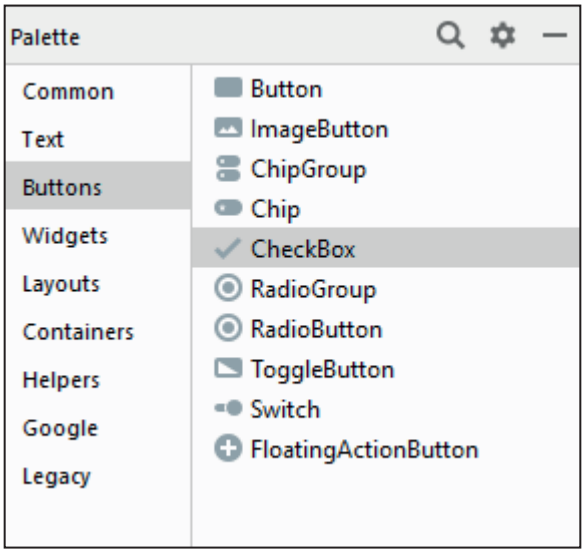
Browse and select the image file from your computer, click **Next** and then select **Import**.

Note: If you click the overflow icon which exists with the resource manager tabs, you will show additional resource types as shown in the following figure:



Adding a Check Box

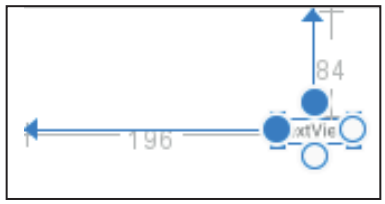
Checkboxes allow users to select one or more options from a set of options. You can drag-and-drop check box from the palette panel to your activity interface.



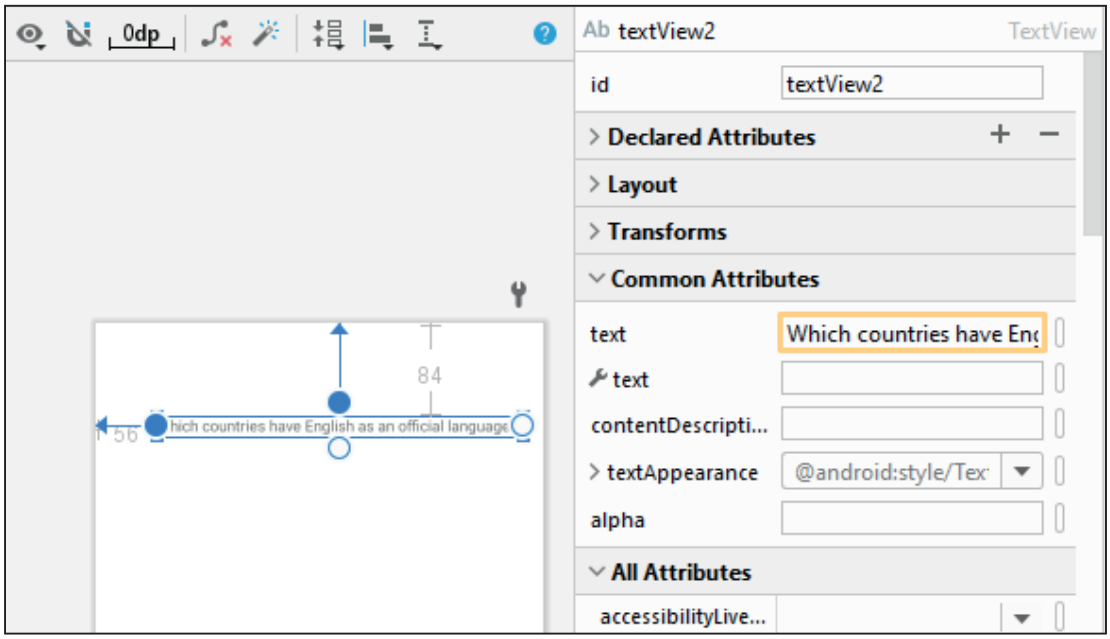
The following example displays how you can add a checkbox to your activity layout and define how its code can be configured:

1- Delete any previous objects in your layout.

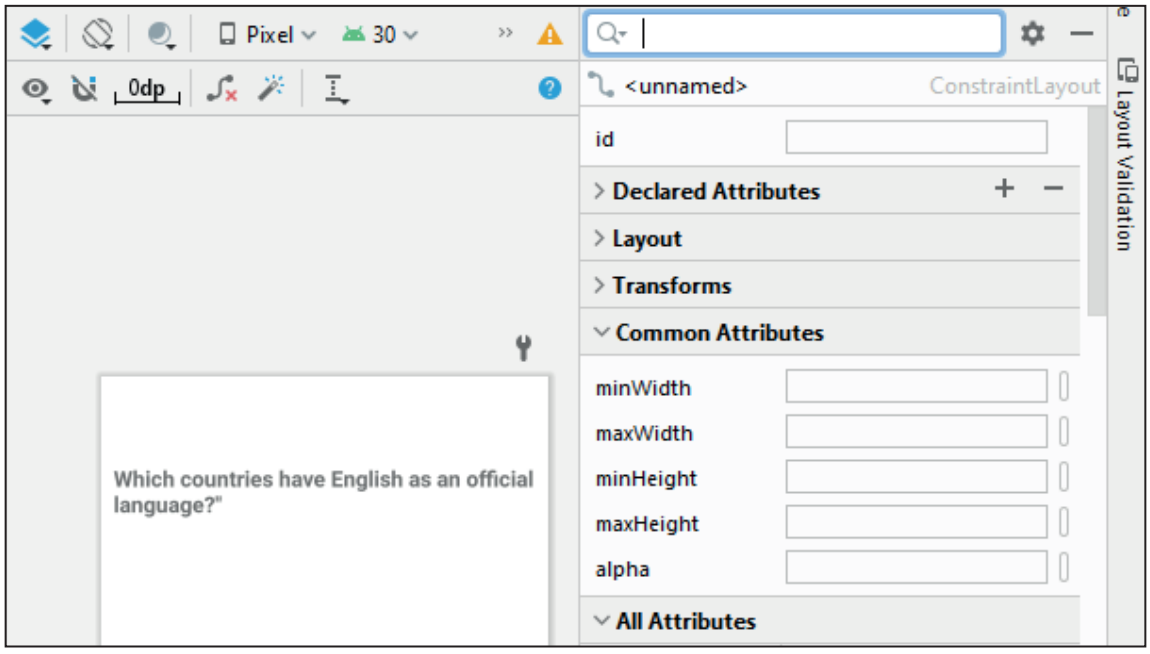
2- From the palette panel, drag and drop **TextView** at the middle top of your activity and set its constraints (top and left margins) as illustrated in the following figure:



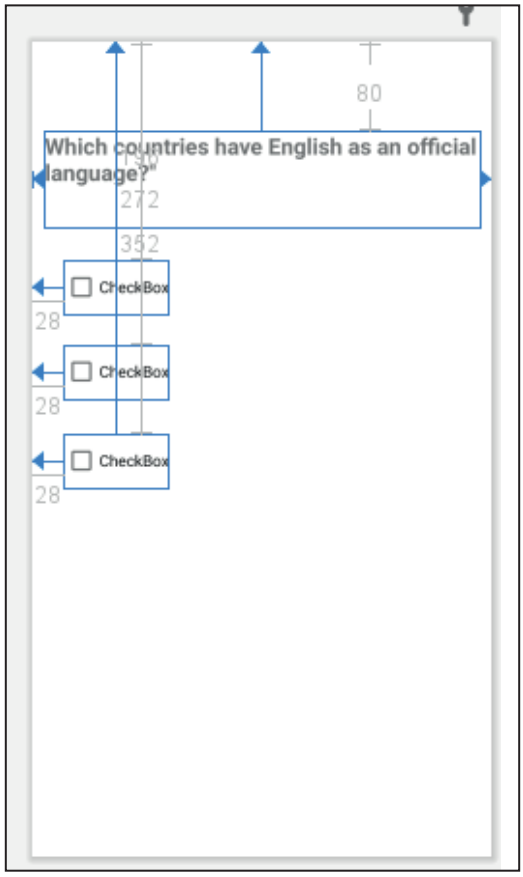
3- Enter the attribute value for "**text**" to be: "**Which countries have English as an official language?**" as illustrated in the following figure:



4- Change the **textSize** attribute value to **20sp** and **textStyle** attribute value to **Bold**. Resize the text box border to show the entire question on the activity layout. Also, set the text box margins. The activity layout will be as follows.



5- From the Palette window, click **Buttons** in the left panel. Drag the **CheckBox** into the activity layout three times, and then set the constraints of these checkboxes (top and left margins) as illustrated in the following figure.



6- Edit the "**text**" attributes value of each checkbox to have the following values: **USA**, **Canada** and **China** respectively. Then the run app result will be as follows.

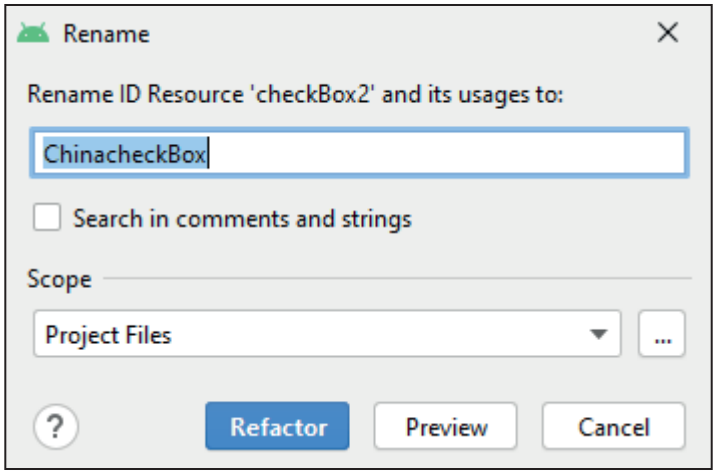
Now, it's time to configure the code of each checkbox, where the code workflow will be as follows.

a) When the app user checks the **USA** checkbox, a textbox with the message "**USA: Correct Answer**" will be displayed. This textbox will be added later.
b) When the app user checks **Canada** checkbox, a textbox with the message "**Canada: Correct Answer**" will be displayed.
c) When the app user checks China checkbox, a textbox with the message "**China: Wrong Answer**" will be displayed.

7- Change the **ID** attribute of each checkbox using the **Attributes** console to the following.

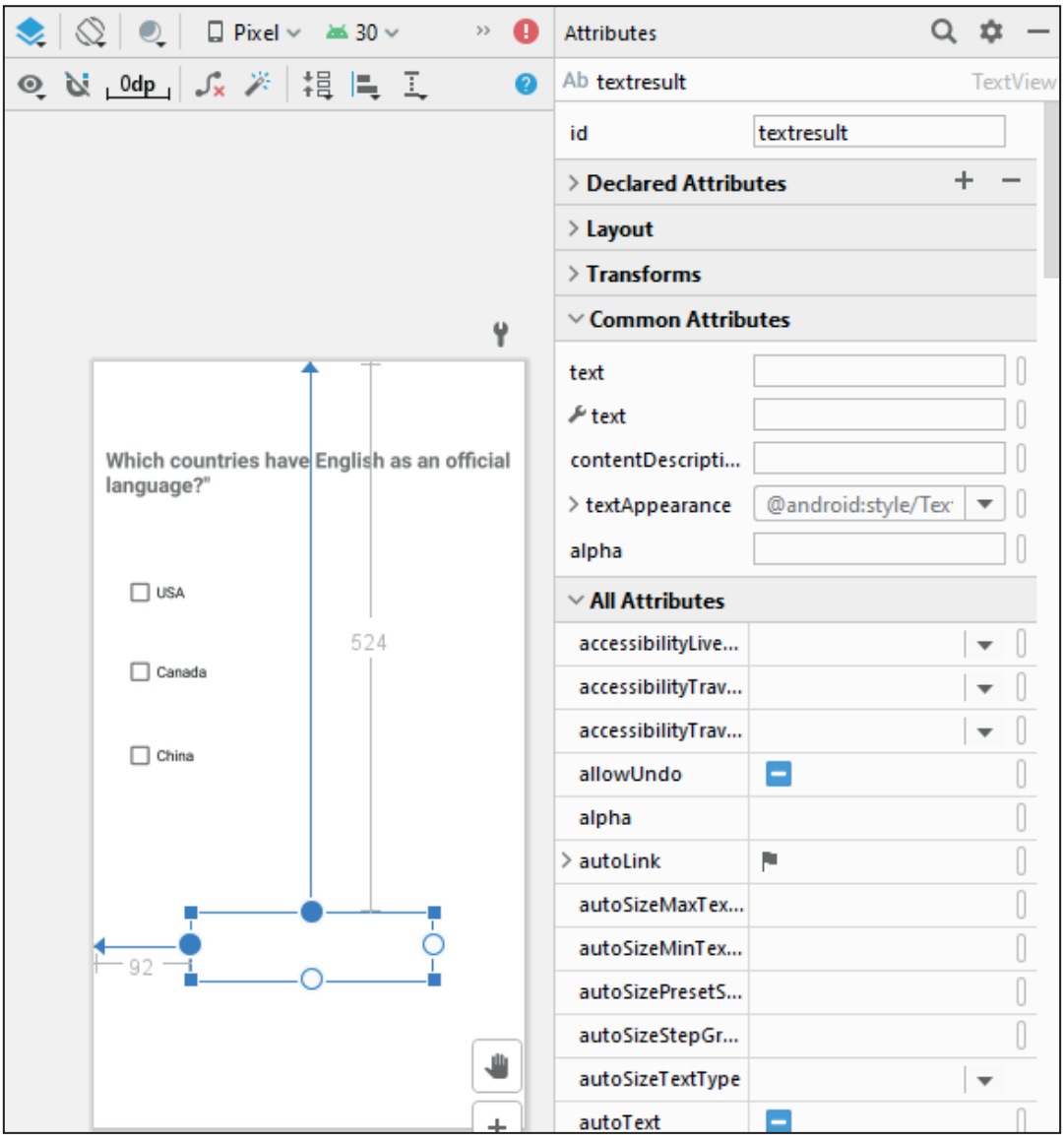
Checkbox	ID
USA	USAcheckBox
Canada	CanadacheckBox
China	ChinacheckBox

If you press Enter key to apply the rename of the checkbox ID, you will get the following dialog box, just click **Refactor** to apply the new id.



The purpose of this step is only to explain the codes which will be written in the next steps. You should understand first how the checkbox in Android generally works, and how you can use it to implement an action or get a result. Normally, checkbox is used with IF or When methods (not clear)Checking the checkbox means that the IF condition is true and a specific action will be done; for example, a flashing text, a calculation operation, sending specific information to a database, etc...

8- Add a **TextView** to **activity_main.xml** (Design mode) using drag-and-drop technique and name the **id** of this **TextView**: **textresult**. This text box will be used later to show the result message produced when selecting a check box.
Set it top and left constraints, increase the size of this text to have enough size to appear the messages and remove the default **Text** attribute value as illustrated in the following figure.



You can write either the IF or When statement. For example, you can use a statement to write the text message" **USA: Correct Answer**" in your activity if you clicked the check box with the id **USAcheckBox** as illustrated in the code below:

```
if (USAcheckBox.isChecked) textresult.text="USA: Correct Answer"
```

This **if statement** will be added to the three choices within a function. In this example, you can call it **onClicked** or any other name and the code will be as follows.
Don't type this code in the MainActivity.kt file now, we will do that in the next steps.

```
fun onClicked(view:View){
if (USAcheckBox.isChecked) textresult.text="USA: Correct Answer"
if (CanadacheckBox.isChecked) textresult.text="Canada:Correct Answer"
if (ChinacheckBox.isChecked) textresult.text="China: Wrong Answer"}
```

Remember that all palette like Check box, Radio button and other items in the palette panel are views (classes) and each one of them is a child class to the parent class **View**. That's why we used the expression **view:View** in the above code.

9- These checkboxes by default will not respond to any click until adding the function "**onClicked**" to their XML tags in the XML activity file:**activity_main.xml**. The attribute to be added to each checkbox is the following:

```
android:onClick="onClicked"
```

When you add this configuration, you will get a red underline for each of these added lines. This error will be corrected automatically when you create the **onClick** function in the **MainActivity** file as you see later. The **activity_main.xml** file be as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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:id="@+id/textView2"
        android:layout_width="389dp"
        android:layout_height="87dp"
        android:layout_marginTop="80dp"
        android:text='Which countries have English as an official
language?"'
        android:textSize="20sp"
        android:textStyle="bold"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```

```
    <CheckBox
        android:id="@+id/USAcheckBox"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="28dp"
        android:layout_marginTop="196dp"
        android:text="USA"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        android:onClick="onClicked"/>

    <CheckBox
        android:id="@+id/ChinacheckBox"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="28dp"
        android:layout_marginTop="352dp"
        android:text="China"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        android:onClick="onClicked"/>

    <CheckBox
        android:id="@+id/CanadacheckBox"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="28dp"
        android:layout_marginTop="272dp"
        android:text="Canada"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        android:onClick="onClicked"/>

    <TextView
        android:id="@+id/textView3"
        android:layout_width="231dp"
        android:layout_height="65dp"
        android:layout_marginStart="92dp"
        android:layout_marginTop="524dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

10- Before starting with typing the Kotlin code, check the **build.gardle** (Module:Lesson05) file that has the Kotlin plugin. Its configuration must be as illustrated in the following figure, and after adding the:

id 'kotlin-android-extensions', click the **Sync Now**

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-android-extensions'
}
```

11- Go to the **MainActivity** to create the **onClicked** function as illustrated in the gray highlighted part of the following code:

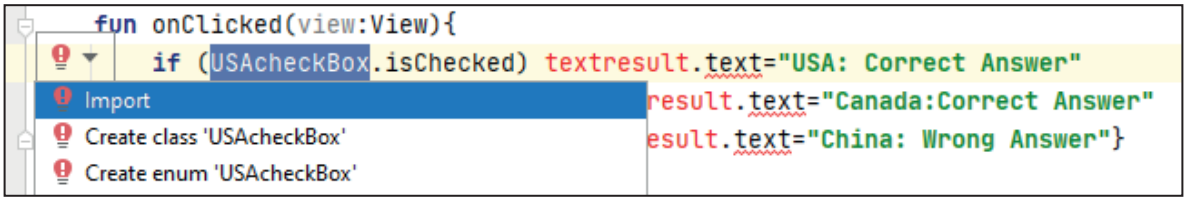
```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
    fun onClicked(view:View){}
}
```

The `view:View` => to declare the view data type is **View**

12-The full function code will be as follows:

```
package com.androidatc.lesson05
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View

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

fun onClicked(view:View){

if (USAcheckBox.isChecked) textresult.text="USA: Correct Answer"
if (CanadacheckBox.isChecked) textresult.text="Canada:Correct
Answer"
if (ChinacheckBox.isChecked) textresult.text="China: Wrong Answer"}

}
```

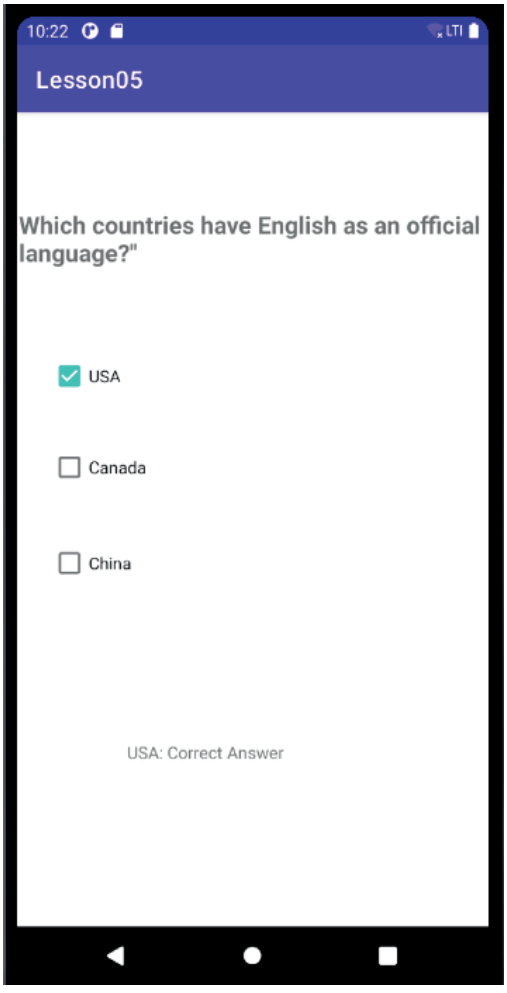
To import the objects in the layout, click on the red underline, click the red lamp, and then select **Import** as illustrated in the following figure:



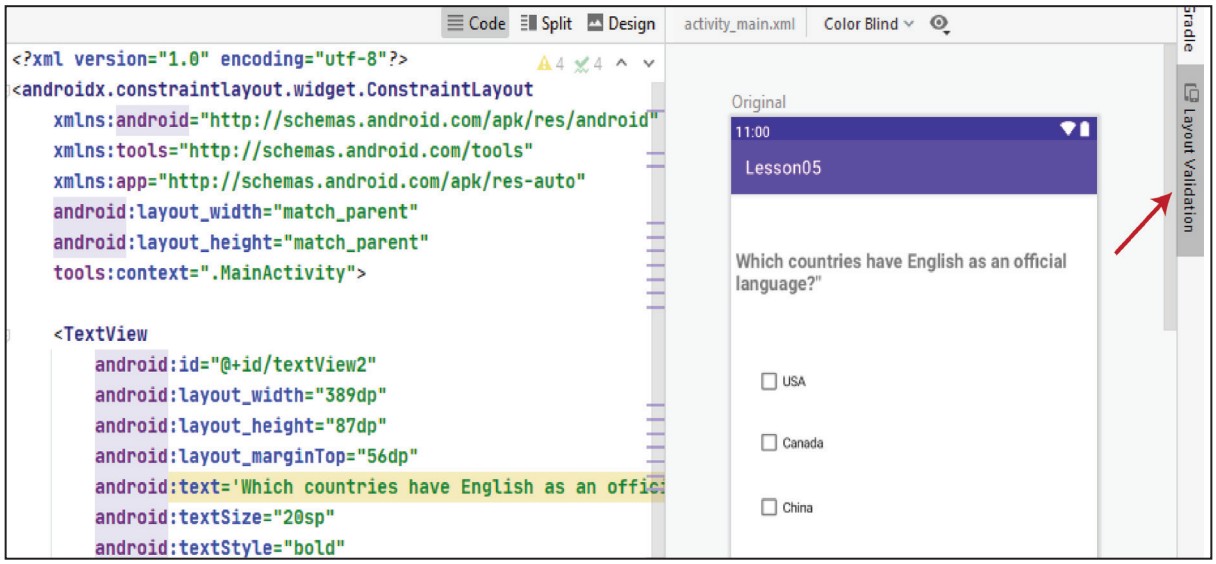
The following line will be added at the top of your code:

```
import kotlinx.android.synthetic.main.activity_main.*
```

13- Click **Run** to run your app and test its operation. The result will be as follows:



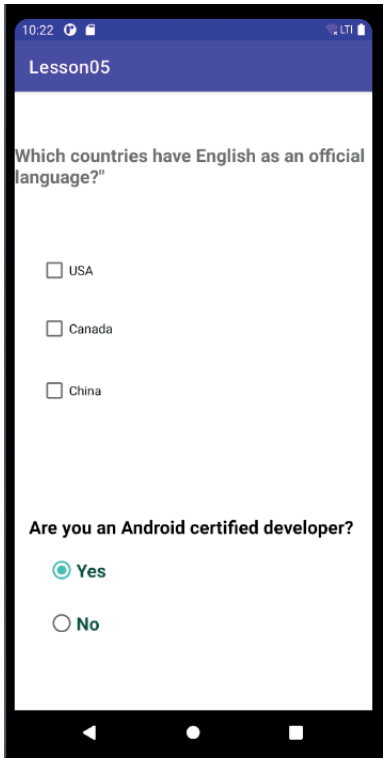
You may work on editing the **activity_main.xml** file and see the effect the changes in the XML code live on the app interface if you open this activity XML file in the Code mode and click the **Layout Validation** tab as illustrated in the following figure:



Adding a Radio Button

In Android development, radio buttons are mainly used together in a **RadioGroup**. By using a radio group, you can only check one radio button from a group of radio buttons which belong to the same radio group. Without Radio Group you can select all the radio buttons simultaneously.

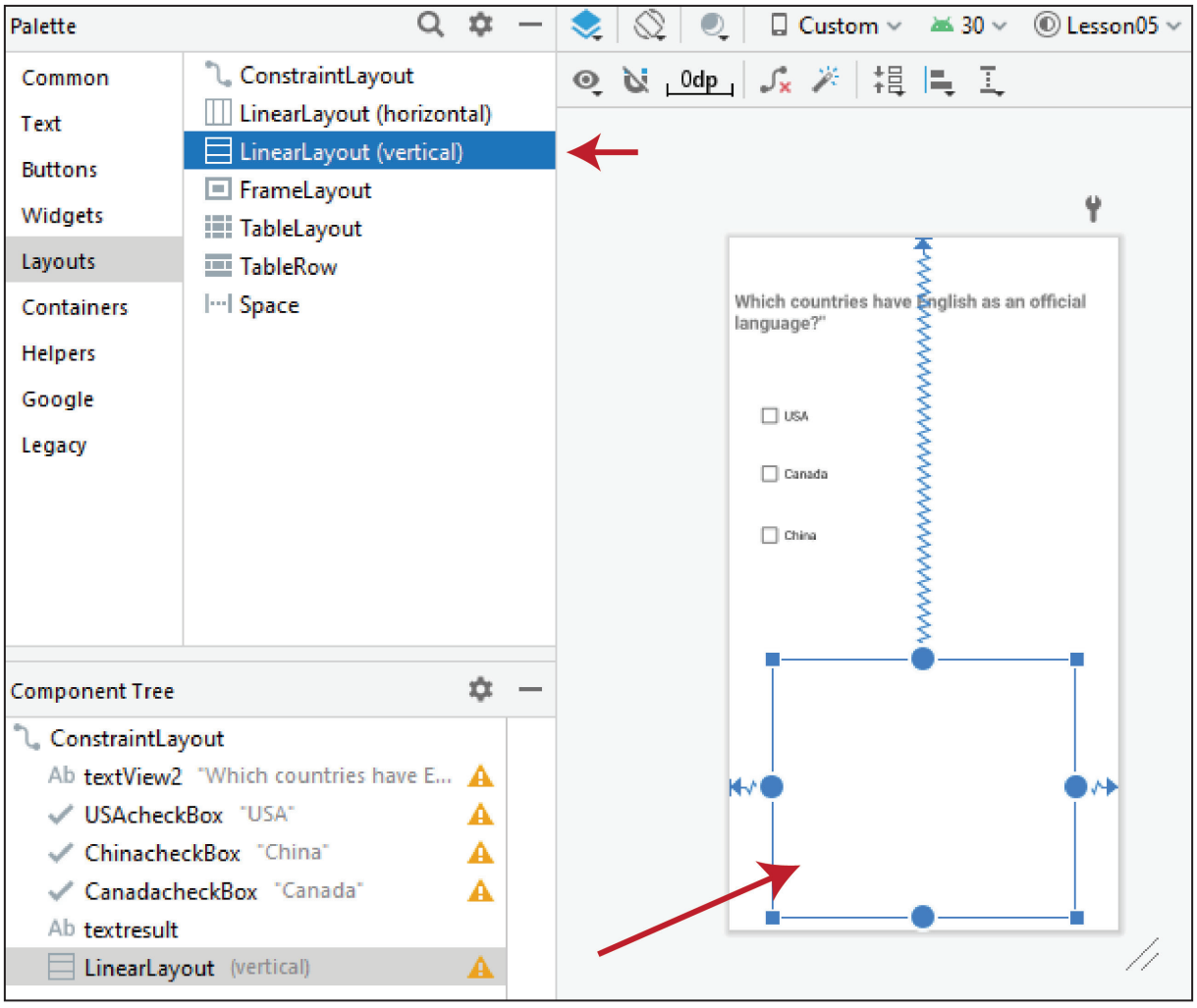
In the following example, you will create a radio button group that consists of two radio buttons and get a certain result based on the option which you will select. You will continue using the same app you worked on for checkbox before, and at the end you will get the following design:



To implement this design, first you should work on the layout design (activity_main.xml) and then the code (MainActivity.kt). To do that, follow the steps below:

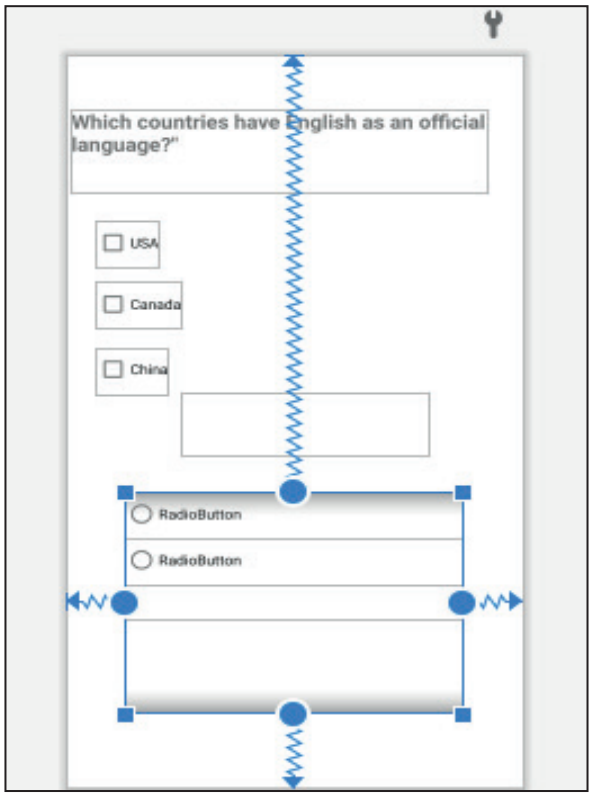
Before you start adding the components of this radio button groups, it is recommended to create all the radio button group components inside a linear layout. A **Linear Layout** is a layout that arranges other views (radio buttons or others) either horizontally in a single column or vertically in a single row. You will learn more in details about linear and other types of layout in the next lesson.

1- Now, to add the two radio buttons, you should add them inside a Linear layout; therefore, add a Linear Layout to your activity first using a drag and drop for the palette panel as illustrated in the following figure and set its constraints to have a similar location and size as illustrated in the figure below:

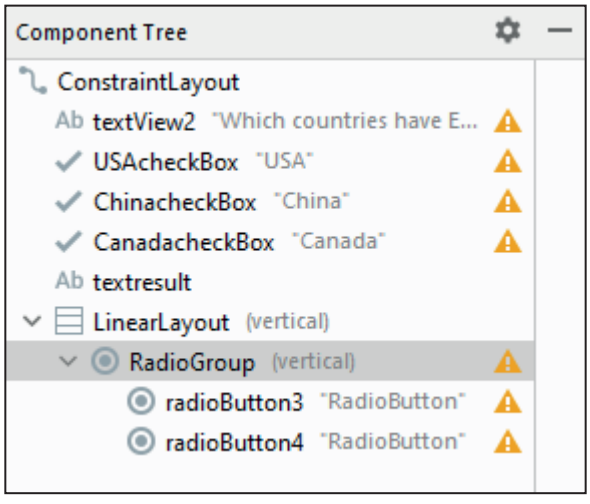


2- Drag from the palette panel (Buttons) a **RadioGroup** inside the square of the Linear layout.

3- Drag two **RadioButton** in the Linear Layout, and you should get the following figure:



Check the component Tree panel which displays that the **RadioGroup** tag which is a parent of the two radio buttons as illustrated in the figure below:



2- Change the "**id**" and "**text**" attribute values for the two Radio Buttons to have the following values:

id	text
radioYes	Yes
radioNo	No

When you change the **id** attribute value, click **Refactor** to replace the previous id with the new one in all this app project.

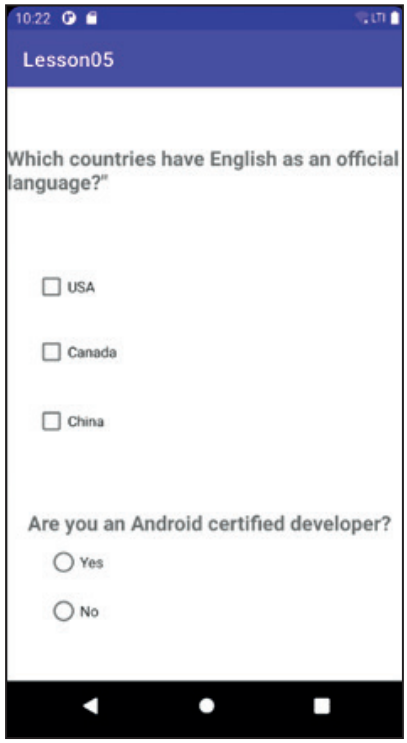
3- **Stop**, then **Run** your app to check if you can select one radio button choice only. You should get a figure similar to the following:

4- From the Palette panel add **TextView** to the activity layout upper the radio buttons, set its constraints, change its **text** attribute value to: "**Are you an Android certified developer?**", change its **textSize** attribute value to: 20sp, and **textStyle** to Bold as illustrated in the following figure:

5- From the Palette panel, add **TextView** to the activity layout directly below the radio buttons, set its constraints (top and left margins), change its **id** attribute value to **textRadioAnswer**, and delete the default **text** attribute value as illustrated in the figure below:

This **TextView** widget will be used later to display the result of the radio button selection when you run your app.

6- Stop, then run and test your app. You should get the following:



7- To write the code for this part (radio buttons), go to **MainActivity** file, add the **onSelect** function to your code, it is similar to the same code which you have used for checkbox before with some changes. Here you can name the function **onSelect** and use any name you want. The full code is as follows:

```
package com.androidatc.lesson05
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
    fun onClicked(view:View){
        if (USAcheckBox.isChecked) textresult.text="USA: Correct
Answer"
        if (CanadacheckBox.isChecked) textresult.
text="Canada:Correct Answer"
        if (ChinacheckBox.isChecked) textresult.text="China: Wrong
Answer"}

    fun onSelect(view:View){
        if (radioYes.isChecked)textRadioAnswer.text="Yes: I am
Android Certified Developer."
        if (radioNo.isChecked)textRadioAnswer.text="No , not
yet."}

}
```

8- Now, you should link this **onSelect** function with the **RadioButton** tags in the XML file. Open **activity_main.xml** and add the gray highlighted attributes to the two radio buttons:

```
<RadioGroup
    android:layout_width="match_parent"
    android:layout_height="133dp">

    <RadioButton
        android:id="@+id/radioYes"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Yes"
        android:onClick="onSelect"/>
```

```
<RadioButton
        android:id="@+id/radioNo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="No"
        android:onClick="onSelect"/>
</RadioGroup>
```

9- **Run** your app to test the radio buttons. You should get the following results: