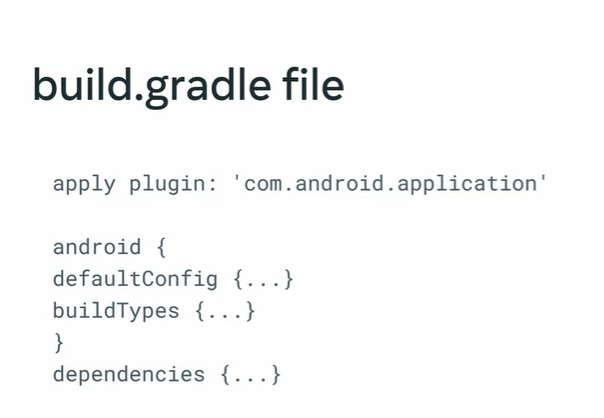
MAIN ACTIVITY

In Android, an activity, which is one thing a user can do, is an entry point for interacting with the user. This is where the action happens because activities allow the user to interact with your app. An Android app is usually made of multiple activities, sometimes called screens, that together form the user experience. These activities are represented by an activity class whose function is to respond to user input. Each activity class contains a layout that holds different pieces of a user interface or UI together so that the user can interact with an app. This means that the activity class is the gateway through which a user can interact dynamically with an Android app's UI. One activity in an app will be specified as the main activity. This is the entry point from which Android will run it's program as it's the first screen that appears when a user launches an app. When you create a new Android project using the main activity template in Android Studio, a file called MainActivity.kt is generated with some initial code alongside other generated files and folders.

GRADLE

With Gradle, the Android project is managed through a set of build configuration files. These define how a project is to be developed, what dependencies need to exist for the project to build and run successfully and what the result or results of the compilation process should be. To start using Gradle in your app project, you don't need to install anything as it comes with Android Studio by default. When you click the run button on the Android Studio toolbar, this action automatically triggers Gradle activity, which in turn starts building the project. After a successful operation, your app starts running on an emulator or a physical device connected to the computer. For every Android project, two build.gradle files are generated. The first build.gradle settings apply to every module in the entire project, while the second set of project-level or module-level settings applies to only the app module. A module is a collection of source files and build settings that allow you to divide your project into different units of functionality. If there were another module within the project, then that module would have its own build.gradle file.



The Android block contains information about your project, such as the minimum operating system or OS version you want your app to use for the compilation.

The default config sub block allows you to specify the minimum OS version, version number, application unique ID and other configuration settings.

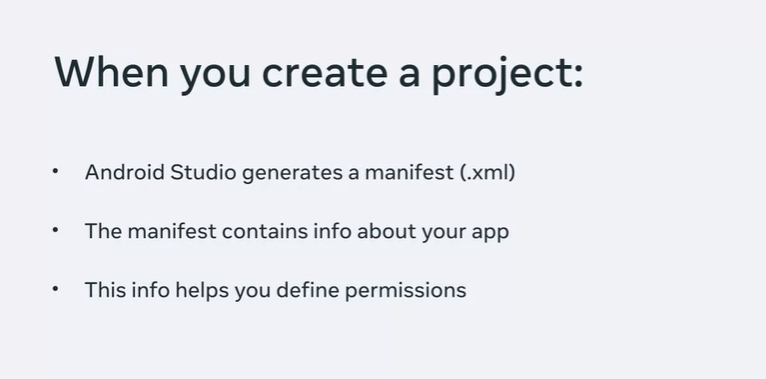
The dependencies block specifies what third-party libraries you wish to include in your app. If a local copy is not available, all specified dependencies are automatically downloaded for you.

GRADLE SYSTEM THROUGH COMMAND LINE TOOL

Gradle system manually through the command line tool. To do so, you'll need to know some Gradle commands, so let's explore some of the popular ones.

1. ./gradlew build. This is used to build a project when you are ready to run the application. A new build directory will be generated if it doesn't exist.
2. ./gradlew clean. You can delete the contents of the build directory already generated using the clean task command.
3. ./gradlew wrapper allows you to see all the available Gradle operations running in the background.

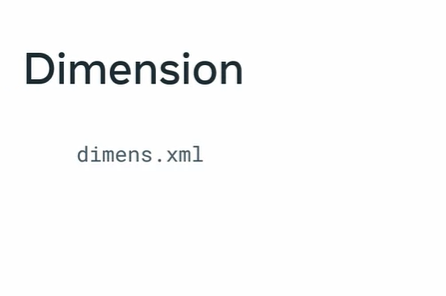
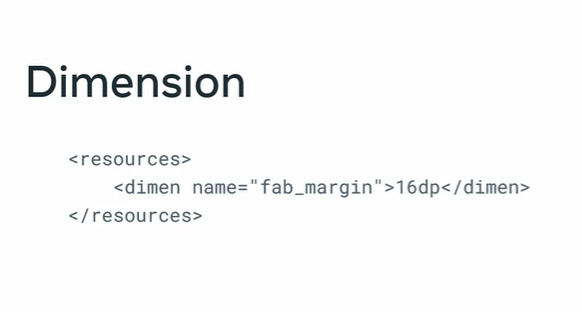
ANDROID MANIFEST

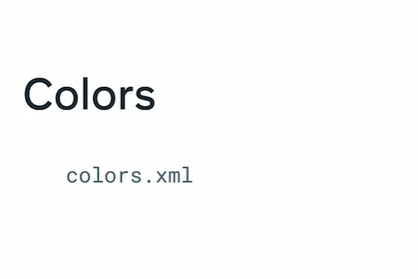


RESOURCE FOLDER

Resources in the form of files and static content that your code uses, such as colors and animations are contained in the Resource folder in Android Studio. The resources in this folder can be referenced anywhere in your application code.

The four are resources

* String: In most cases, you are going to display text in one or more places across your app. This text needs to be stored so that different parts of the app can access it. String resource gives you the capability to define the text in the res/values/strings.xml file. Each entry is a key that represents the ID of the text and a value that is the text itself. For instance, if I want a text to display Congratulations!, on my app, I might add this string to the string resource file called strings.xml.
* Colour: Creating a beautiful user interface or UI, comprises the use of the right combination of elements and colors. If you ever need to use colors across your app, the color resource helps you manage your colors so that they are easily accessible across the app. Colors should be defined within the file called colors.xml.
* Dimension: you will likely use different sizes and dimensions for elements in your app, there's a dimension resource. The dimension resource makes it easy for you to manage all your size dimensions in one place, dimensions should be defined within a dimens.xml file.
* Font: there's the font resource, as you may be able to tell from its name, this folder is used to manage the fonts you use within your entire app project. All your front files should be added to the front directory.



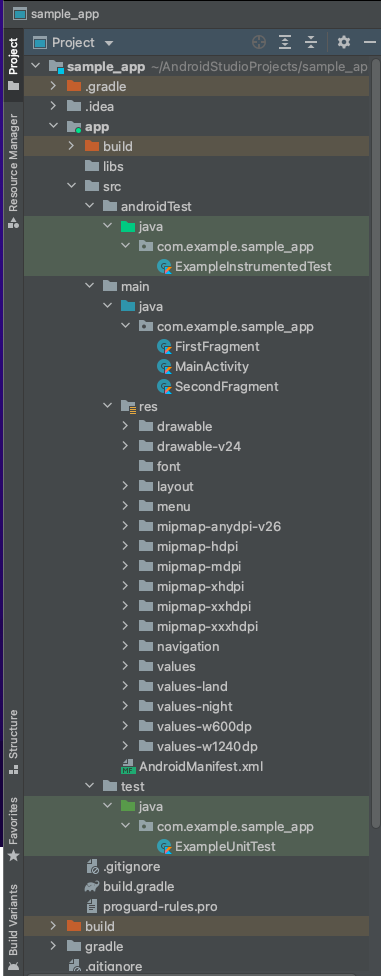
***Project files***

As part of your learning related to project structure in Android Studio, it is essential for you to be aware of some of the key files within this structure and their functions.

To access the entire file structure of a project including all files hidden within folders from the Android View, select Project from the dropdown at the top of the Project window.

Project tree menu for sample appProject tree menu for sample app

Choosing Project View displays many more files and directories.



Project file types

The most important project files and their contents include:

build/

Contains files generated after a project build occurs

libs/

Contains private and 3rd party libraries used within the app

src/

Contains all code and resource files for the module in its subdirectories

androidTest/

Contains code for testing features of the app that runs on an emulator or a physical device

test/

Contains code used for unit testing code, with unit tests executed on the computer

main/

Contains the Android code and resources shared by all build variants for testing and production deployments

AndroidManifest.xml

Describes specific information about your app and each of its components such as Activity

java/

Contains Java and Kotlin-related code files

gen/

Contains the Java files generated by Android Studio, which are required for the app to build successfully (not displayed in the image above)

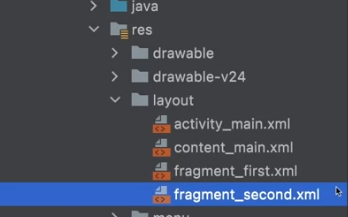
res/

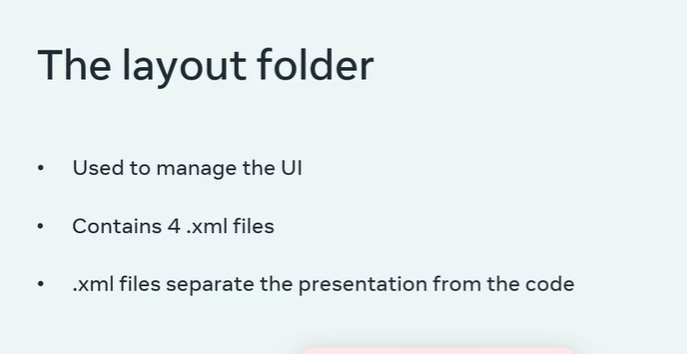
Contains application resources, such as image files, layout files for user interface designs, color files, dimension files and UI string for managing texts used across the app

assets/

Contains files that should be compiled into an .apk file as-is (not displayed in the image above)

You can navigate this directory in the same way as a typical file system using URIs and read files as a stream of bytes using the AssetManager. assets/ is a good location for video files, for example.





***VIDEOVIEW CLASS***

VideoView class

One of the primary uses for mobile devices is to enable the user to be entertained by offering them access to quality content. One key form of content widely used, especially for tablet devices, is video. As part of developing your own video app player in Android Studio, you can use the VideoView class.

What is VideoView class?

VideoView class allows you to play video files in Android.

The Android SDK includes two classes that make video playback implementation on Android devices easy to implement when developing apps. VideoView can be used to display and play a video in an Android app.

Whenever you add this component to the layout of an activity, it provides a surface onto which a video may be played. The following video formats are currently supported on Android:

H.263

H.264 AVC

MPEG-4 SP

VP8

Aside from the surface to display a video, the VideoView class has a wide range of methods that may be called in order to control the playback of video. Some of the more commonly used methods are as follows:

setVideoPath(String path)

This specifies the path (as a string) of the video media to be played. This can be either a remote video file URL or a local video file on the device.

setVideoUri(Uri uri)

This function operates the same way as the setVideoPath() method but takes a URI object as an argument instead of a string.

start()

When this function is used, it starts video playback.

stopPlayback()

This function stops the video playback.

pause()

This function pauses video playback.

isPlaying ()

This function returns a Boolean value that indicates whether a video is currently playing.

What is MediaController?

The MediaController is the user interface to control the video.

Specifically, it is a view that contains media controls such as "Play/Pause", "Rewind", "Fast Forward" and a progress slider. It synchronizes the controls with the state of the MediaPlayer.

The MediaController comes with a default set of controls. If you want to make changes to it, you have to implement the MediaController class programmatically within your Kotlin code.

You can also use ExoPlayer for audio and video streaming in Android apps, but it is not distributed with the Android SDK. To find out more about this, you can check out the additional readings for this lesson.

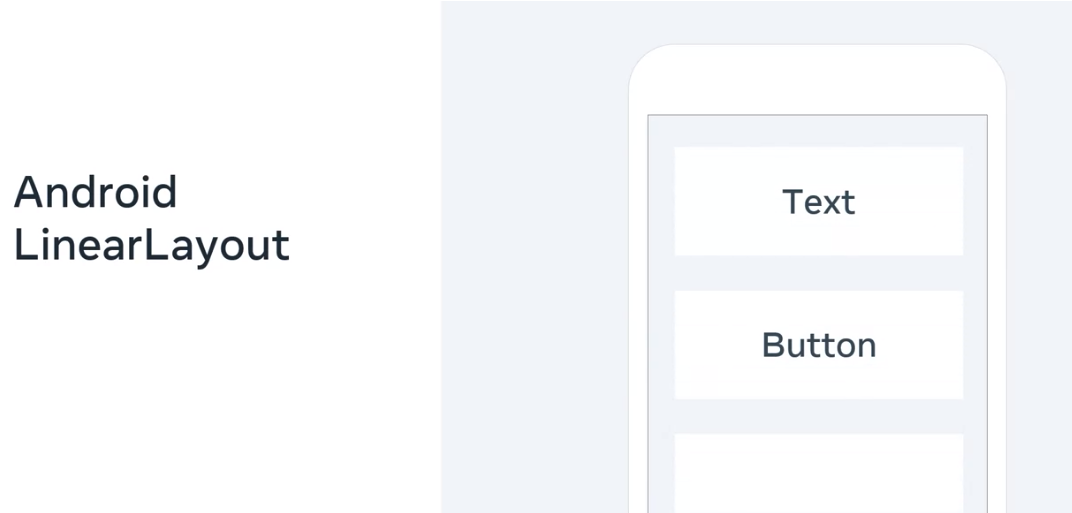
***LAYOUTS***

­­ An Android app contains one or more activities. Each activity is one screen of the app. These screens will contain one or more UI components. The structure of these UI components is defined by layouts. Layouts are comprised of a collection of views and ViewGroup objects.

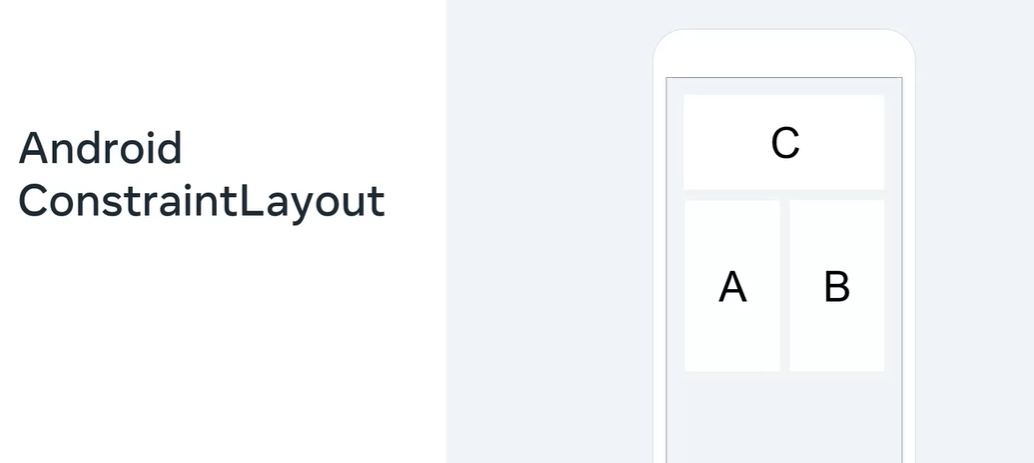
A view is Android's built-in view class. When this is used to create a view, it can be referred to as an object. It represents a rectangular area of the screen and is used for displaying information or content and event handling. Text, images, and buttons are all views on Android. They can be grouped inside another view which can act as a container.

ViewGroup is a special use of the view class because it can contain one or more other views. In other words, ViewGroup is used to define the layout in which views will be arranged on the Android screen.

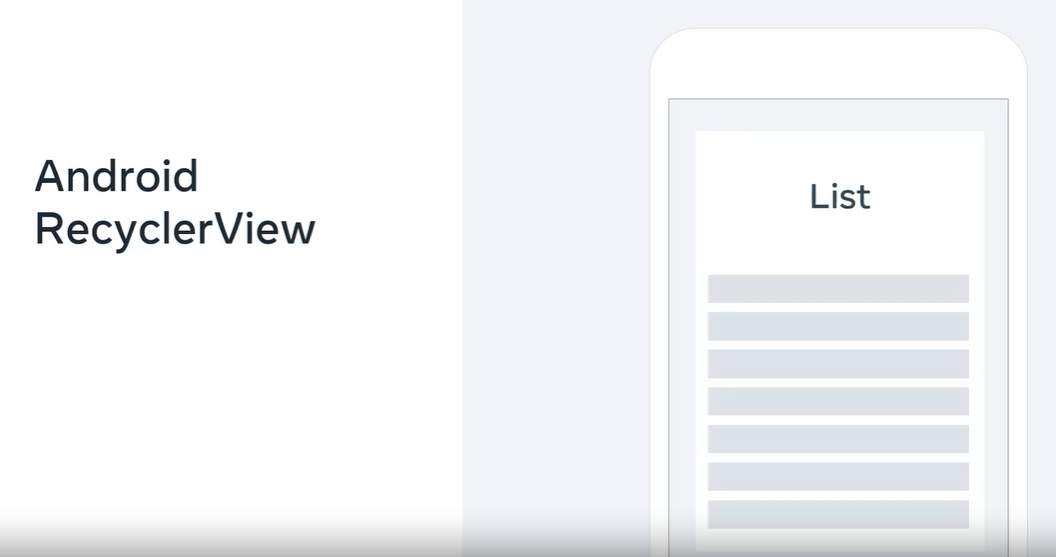
Android linear layout. Linear layout is an extension of a ViewGroup, which is used to render different child view instances, such as text, button and so on. This layout is used by setting the orientation to either a horizontal or a vertical direction, one after the other.



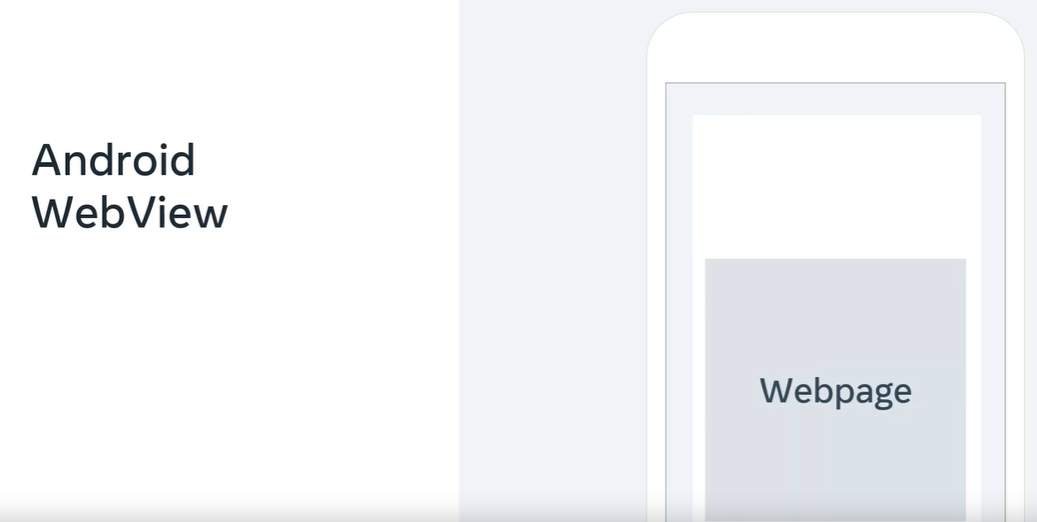
Android constraint layout. Constraint layout is a ViewGroup subclass that enables developers to position and size child views or widgets flexibly using constraints. These constraints make it possible to position a child view relative to other views on the screen. For example, you need to use constraints to constrain child B to the right of child A, child C to the top of child A and B.



Android RecyclerView. This is a ViewGroup that can be used to display sets of data that can be scrolled. The item views inside the RecyclerView are positioned using layout managers. For example, linear layout manager displays items in a vertical or horizontal list and grid layout manager displays items in a grid.



Android WebView. WebView is an extension of a ViewGroup which is used to render webpages as part of an activity layout.



HOW TO CREATE A LAYOUT FILE->

Res folder -> layout

***Layouts***

Part of setting up the screen of your app in Android Studio involves layouts. Read on to learn more about the function and types of layouts in Android Studio.

What is a layout?

In Android, layout defines the user interface (UI) for an app or activity and holds the UI elements that will appear to the user.

Related to layouts are View and ViewGroup.

View

A View is defined as the UI which is used to create interactive UI components such as TextView, ImageView, Label, RadioButton and so on. It is in charge of event handling and drawing. They are generally referred to as "widgets".

ViewGroup

A ViewGroup serves as a parent class for layouts and layout parameters that hold other Views or ViewGroups and define the layout properties. They are generally referred to as "layouts".

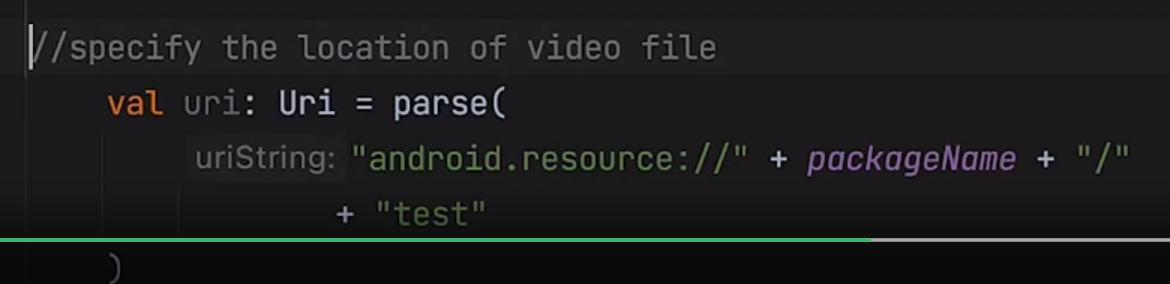
Types of Android Layouts

* LinearLayout: LinearLayout is a ViewGroup subclass used to render child View elements one after the other either horizontally or vertically based on the orientation property specified.
* ConstraintLayout: ConstraintLayout is a ViewGroup subclass used to specify the position of layout constraints for every child View relative to other views on the screen.
* Frame Layout: FrameLayout is a ViewGroup subclass used to specify the position of View elements it contains on the top of each other to display only a single View inside the FrameLayout.
* Table Layout: TableLayout is a ViewGroup subclass used to display the child View elements in rows and columns.
* WebView: WebView is a browser that is used to display the web pages in your activity layout.
* RecyclerView: You can use RecyclerView with LinearLayoutManager to display scrollable lists of items in a single column. You can also use RecyclerView with GridLayoutManager to display a scrollable list of items in a grid view of rows and columns.

FINDING THE ASSESTS->

Android has a built-in class specifically for locating files. It's called URI, which stands for Uniform Resource Identifier. This is essentially the file path and the description of the resource you want to locate. A URI identifies a resource either by location, name, or both. Take note that a resource can be any type of file. It can be an image, video, or music file.

click on the res folder, go to New, and select New Android resource directory.



This code will make a reference to the file because it specifies where to find it. First, it specifies that it is an Android resource. Then it specifies the package name, which in this case is com.example. Lastly, it specifies to retrieve the file test from this directory, make sure that you do not add the file extension. Android automatically recognizes the file as a video file because it's added to the raw folder inside the Android Resource folder.

***Android Views***

The main building block for the user interface (UI) of an app like the video player you are preparing to develop is creating a View.

What are Views?

The View object is created from the View class and occupies a rectangular screen area. It is responsible for the processing of drawings and events.

The View is the base class for widgets that are used to create interactive components of the user UI such as buttons or text fields.

In summary, a View can be considered a rectangle on the screen that displays a certain type of content, like an image, a piece of text or even a button!

Most used Android Views

Some of the most used Android Views include:

* TextView
* EditText
* Button
* ImageView
* ImageButton
* CheckBox
* RadioButton
* RecyclerView
* DatePicker and
* Spinner
* View sizes

A View occupies a rectangle shape, although the rectangle is invisible.

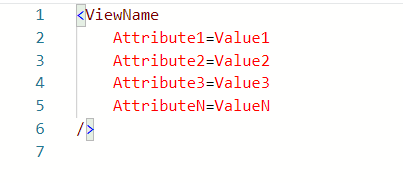
The size of this rectangle can be manually set by specifying the exact size (with appropriate units) or using some predetermined values. These predefined values are match\_parent and wrap\_content.

match\_parent means it will occupy all of the available space available within its parent ViewGroup wrap\_content means it will occupy only as much space as required for its content to display.

XML syntax for creating a View

Now, as you learned previously, to draw anything in your Android application you need to specify it in the design XML files. You will create Kotlin files to add functionality.

Every View in XML has the following format:



* It always starts with an angle bracket, followed by the View name.
* You then write attributes that will determine what this view will look like on the app screen along with a value for the attribute. Each view has its own unique attributes.
* In the end, it is closed by a forward slash and an angle bracket.

So, every View subclass needs to follow this standard so that it can display on the screen of the app. This format is the default standard of XML.

Take note that there are two attributes that are required for every View. These are android:layout\_height and android:layout\_width.

These attributes determine the size of the invisible rectangle created by the view. Using these attributes, you control the size of every view in your Android app.