MOBILE APPLICATION DEVELOPMENT LAB

Prepared By,
Rini Kurian
Assistant Professor
AJCE

Contents

- **Android Fundamentals–**

  Android Introduction, Application structure,Activities and Activity lifecycle, Logcat usage, Views.

- **Basic UI design –**

  Form widgets, Text Fields, Validation of EditText, Layouts, [dip, dp, sip, sp] versus px, validations on various UI controls.

- **Preferences–**

  Shared Preferences, Preferences from xml .

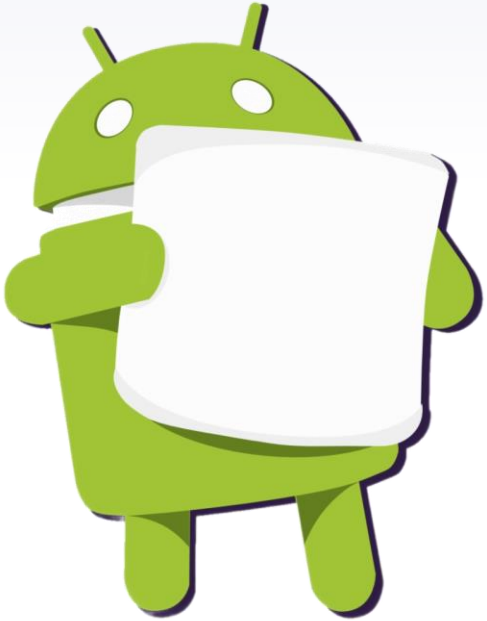
- **Android Menu –**

  Option menu, Context menu, menu from xml, menu via code).

Let's Find Out
All About

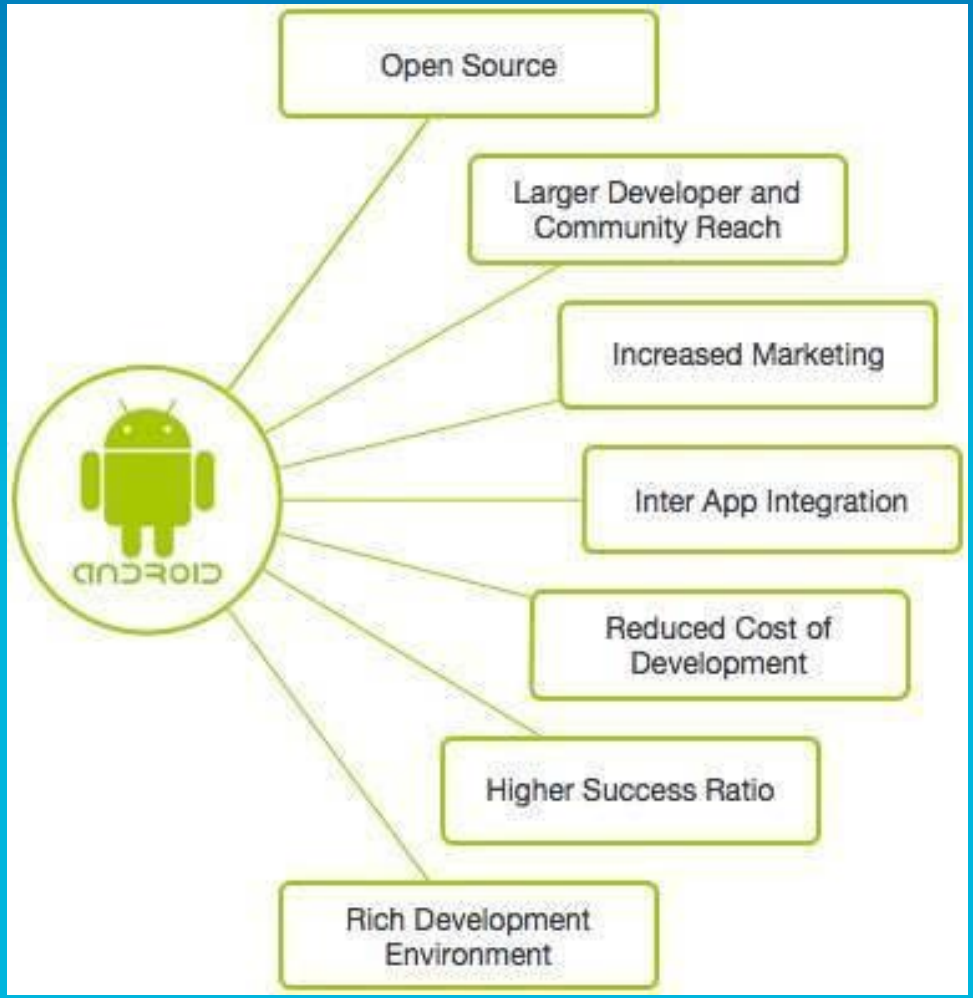
android

What is Android ?



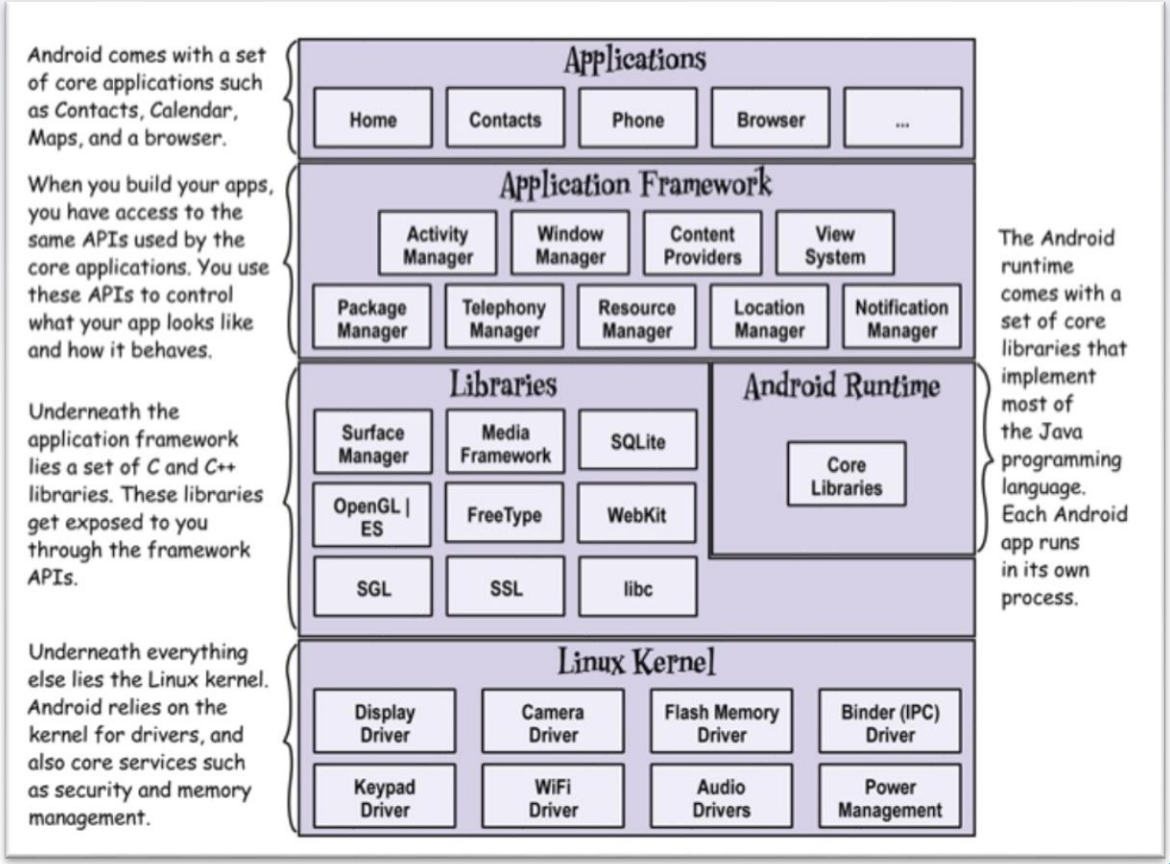
Android OS is an open source, Linux-based mobile operating system that primarily runs on smartphones and tablets.

The Android platform includes an operating system based upon the Linux kernel, a GUI, a web browser and end-user applications .


Open Source
Larger Developer and Community Reach
Increased Marketing
Inter App Integration
Reduced Cost of Development
Higher Success Ratio
Rich Development Environment
android

Android Architecture-Software Stack of Android

- The Android operating system follows a layered architecture approach. All these layers are responsible for different roles and features.
- Android architecture is a software stack of components to support mobile device needs.
- Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram.



Linux kernel:

- Provides a level of abstraction between the device hardware.
- Contains all the essential hardware drivers like camera, keypad, display etc.
- Handles all the things such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

Libraries: On top of Linux kernel there is a set of libraries exposed to developers through Android Application Framework.

- SQLite Library used for data storage and light in terms of mobile memory footprints and task execution.
- WebKit Library mainly provides Web Browsing engine and a lot more related features.
- The surface manager library is responsible for rendering windows and drawing surfaces of various apps on the screen.
- The media framework library provides media codecs for audio and video.
- The OpenGl (Open Graphics Library) and SGL(Scalable Graphics Library) are the graphics libraries for 3D and 2D rendering, respectively.
- The FreeType Library is used for rendering fonts.

Android Libraries: This category encompasses those Java-based libraries that are specific to Android development.

- android.app – Provides access to the application model and is the cornerstone of all Android applications.
- android.content – Facilitates content access, publishing and messaging between applications and application components.
- .android.text – Used to render and manipulate text on a device display.
- android.view – The fundamental building blocks of application user interfaces.
- android.widget – A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.

Android Runtime

Provides a key component called Dalvik Virtual Machine which is a kind of Java Virtual Machine specially designed and optimized for Android.

The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language.

The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine.

The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

Applications

You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games etc.

Application Framework

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

The Android framework includes the following key services −

- Activity Manager − Controls all aspects of the application lifecycle and activity stack.
- Content Providers − Allows applications to publish and share data with other applications.
- Resource Manager − Provides access to non-code embedded resources such as strings, color settings and user interface layouts.
- Notifications Manager − Allows applications to display alerts and notifications to the user.
- View System − An extensible set of views used to create application user interfaces.

Android Application Components

Activities

Dictates the UI and handles the user interaction to the smartphone screen.

Broadcast Receivers

Handles the communication between Android Operating System and the Application.

Services

Handles the background processing associated with an application.

Content

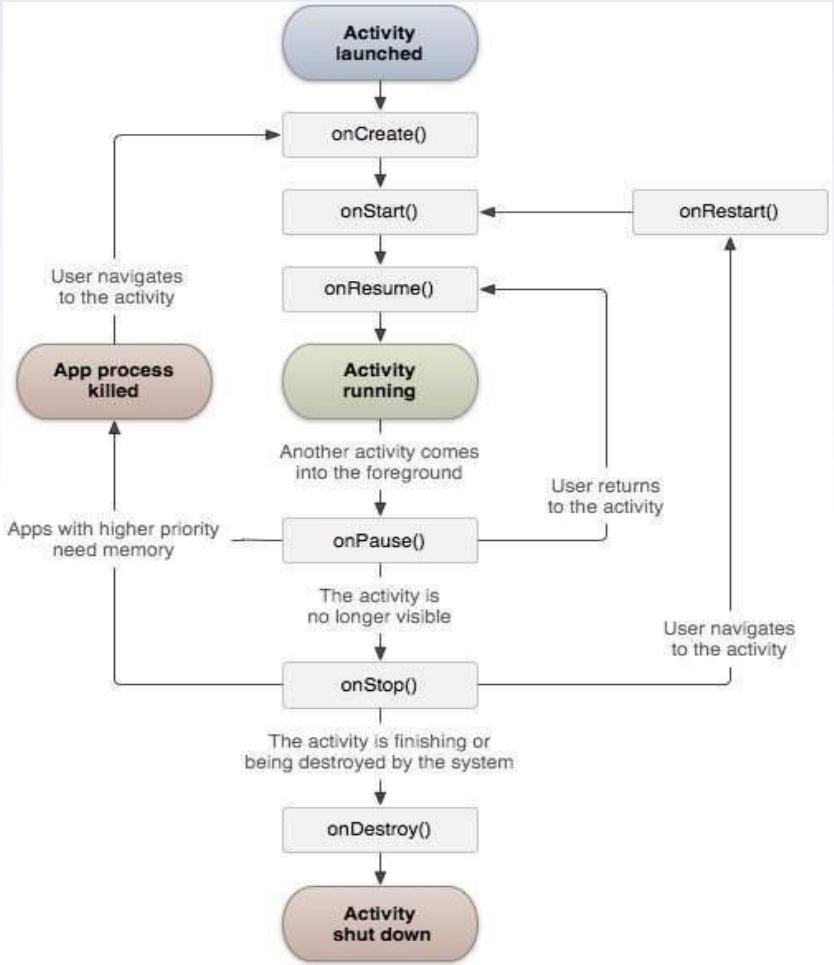
Handles the data and database management issues.

Activities

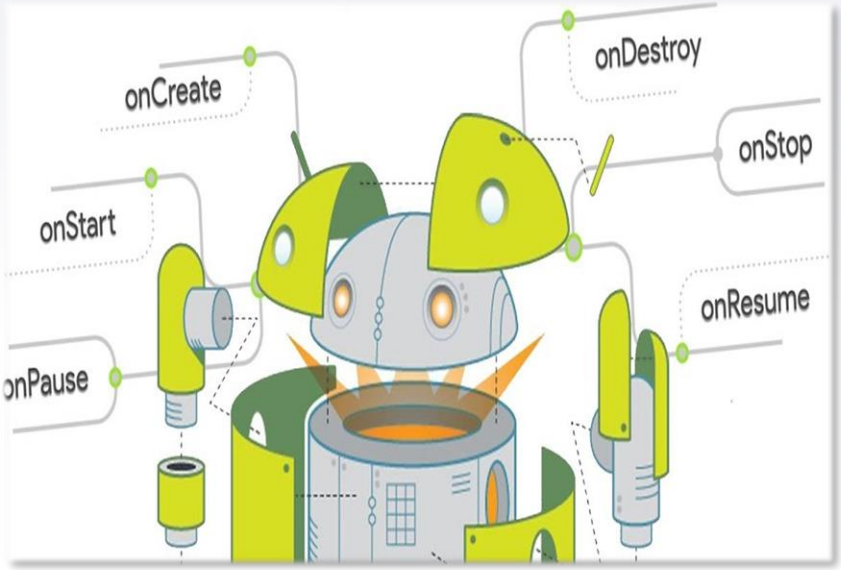
- An activity represents a single screen with a user interface, in-short ,Activity performs actions on the screen.
- If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.
- An activity is implemented as a subclass of Activity class as follows –

```
public class MainActivity extends Activity {
}
```

Android system initiates its program with in an Activity starting with a call on *onCreate()* callback method



Method	Description
onCreate	called when activity is first created.
onStart	called when activity is becoming visible to the user.
onResume	called when activity will start interacting with the user.
onPause	called when activity is not visible to the user.
onStop	called when activity is no longer visible to the user.
onRestart	called after your activity is stopped, prior to start.
onDestroy	called before the activity is destroyed.

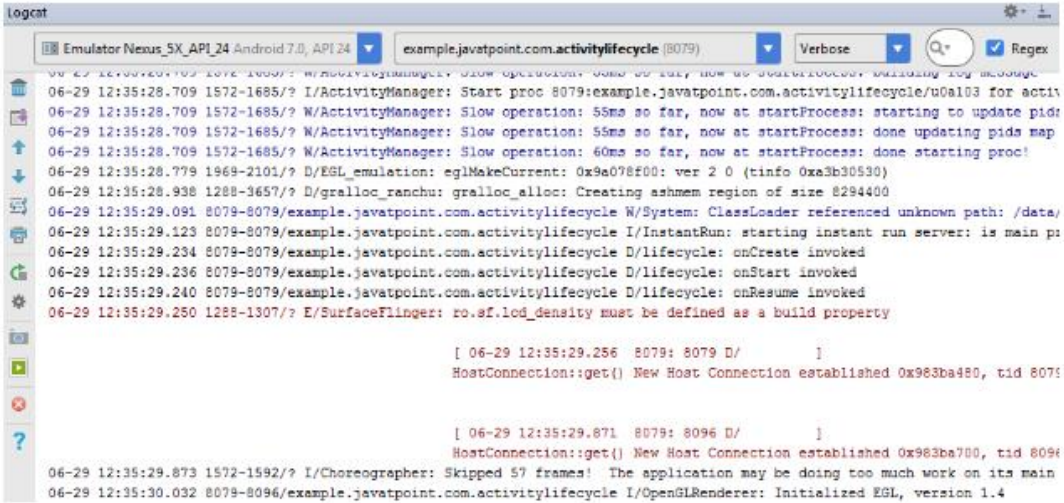


```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.javatpoint.com.activitylifecycle.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>
```

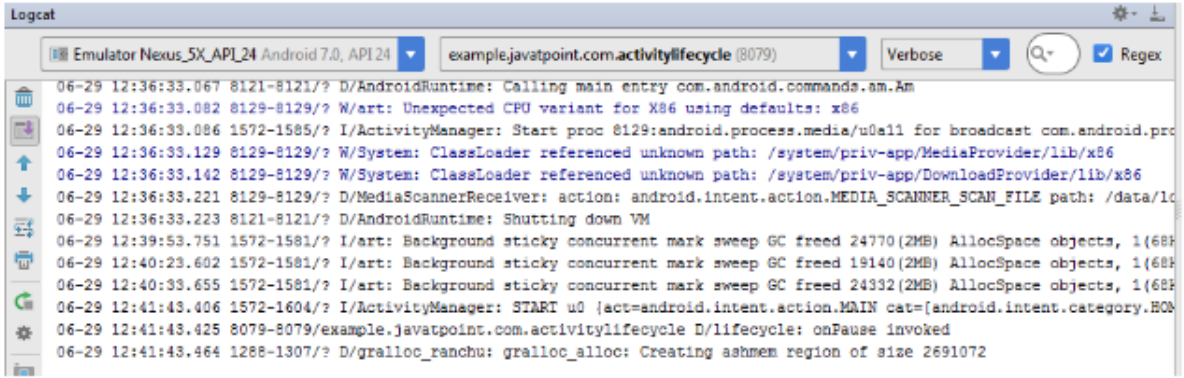
File: MainActivity.java

```
package example.javatpoint.com.activitylifecycle;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Log.d("lifecycle","onCreate invoked");
  }
  @Override
  protected void onStart() {
    super.onStart();
    Log.d("lifecycle","onStart invoked");
  }
  @Override
  protected void onResume() {
    super.onResume();
    Log.d("lifecycle","onResume invoked");
  }
  @Override
  protected void onPause() {
    super.onPause();
    Log.d("lifecycle","onPause invoked");
  }
  @Override
  protected void onStop() {
    super.onStop();
    Log.d("lifecycle","onStop invoked");
  }
  @Override
  protected void onRestart() {
    super.onRestart();
    Log.d("lifecycle","onRestart invoked");
  }
  @Override
  protected void onDestroy() {
    super.onDestroy();
    Log.d("lifecycle","onDestroy invoked");
  }
}
```

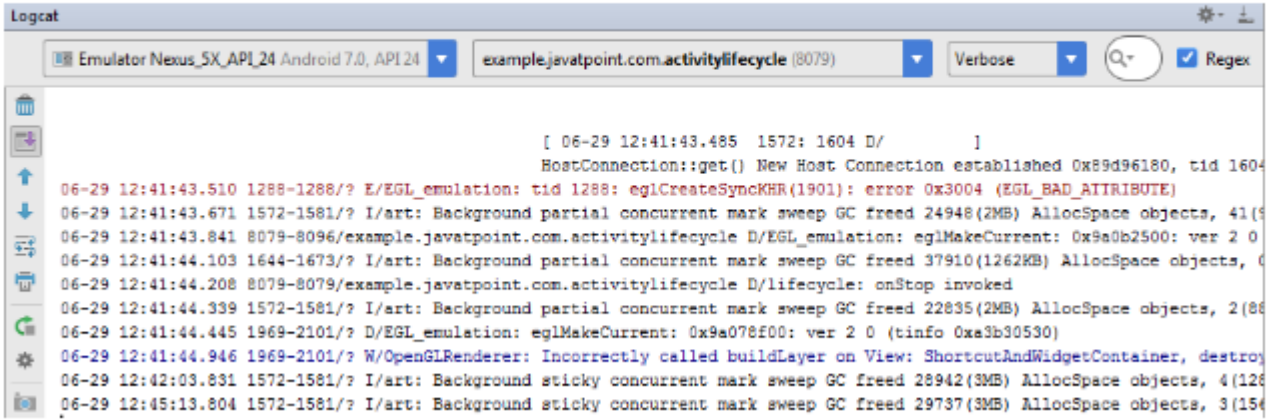
Now see on the logcat: onCreate, onStart and onResume methods are invoked.



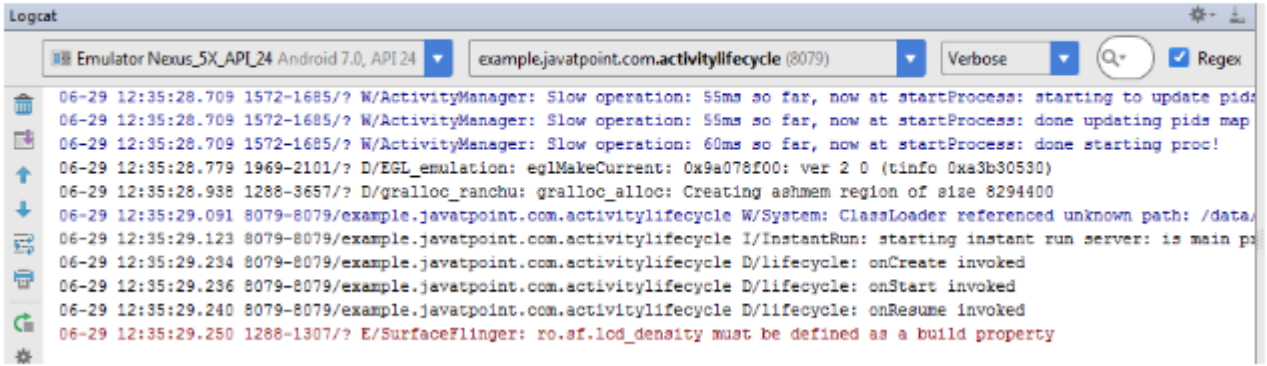
Now click on the HOME Button. You will see onPause method is invoked.



After a while, you will see onStop method is invoked.



Now see on the logcat: onRestart, onStart and onResume methods are invoked.



Services

- A service is a component that runs in the background to perform long-running operations.
- For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.
- A service is implemented as a subclass of Service class as follows –

```
public class MyService extends Service {
}
```

Broadcast Receivers

- Broadcast Receivers simply respond to broadcast messages from other applications or from the system.
- For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.
- A broadcast receiver is implemented as a subclass of BroadcastReceiver class and each message is broadcaster as an Intent object.

```
public class MyReceiver  extends  BroadcastReceiver {
  public void onReceive(context,intent){}
}
```

Content Providers

- A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the *ContentResolver* class. The data may be stored in the file system, the database or somewhere else entirely.
- A content provider is implemented as a subclass of ContentProvider class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class MyContentProvider extends  ContentProvider {
  public void onCreate(){}
}
```

Additional Components

S.No	Components & Description
1	**Fragments-** Represents a portion of user interface in an Activity.
2	**Views**- UI elements that are drawn on-screen including buttons, lists forms etc.
3	**Layouts**- View hierarchies that control screen format and appearance of the views.
4	**Intents**- Messages wiring components together.
5	**Resources**- External elements, such as strings, constants and drawable pictures.
6	**Manifest**- Configuration file for the application.

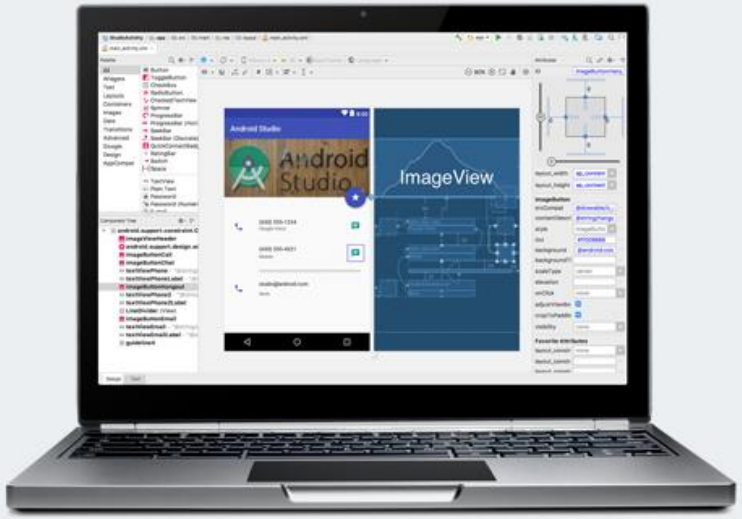
Android Studio

The Official IDE for Android

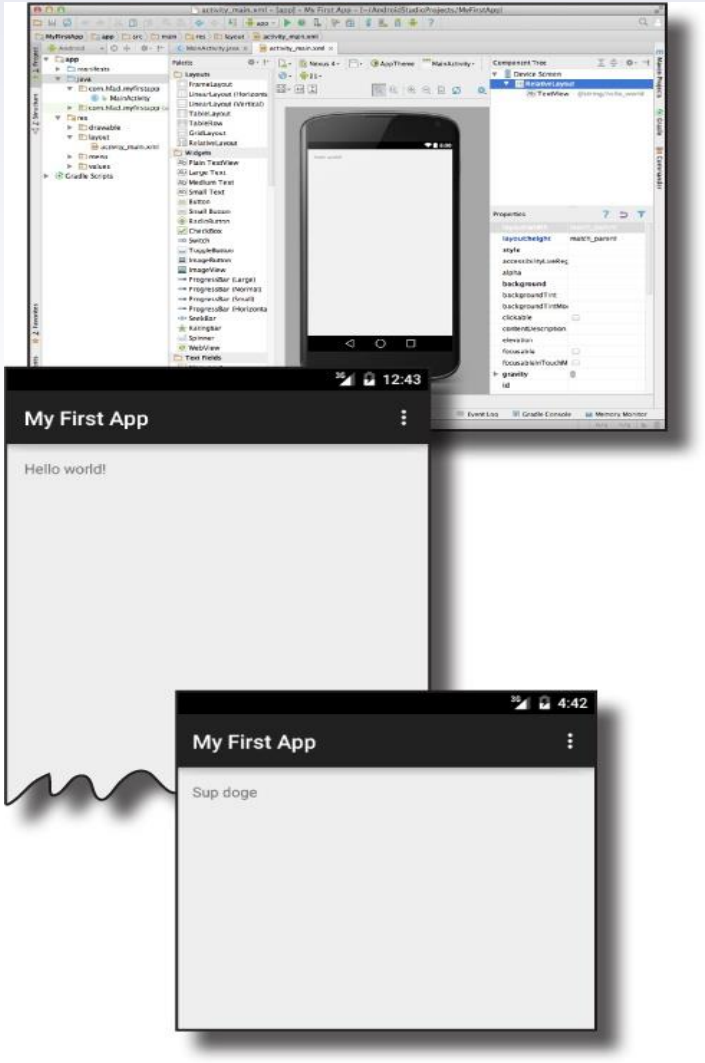
Android Studio provides the fastest tools for building apps on every type of Android device.

World-class code editing, debugging, performance tooling, a flexible build system, and an instant build/deploy system all allow you to focus on building unique and high quality apps.

DOWNLOAD ANDROID STUDIO
3.0.1 FOR WINDOWS (683 MB)



Create Your first Android Application



1. Set up a development environment.

We need to install Android Studio, which includes all the tools you need to develop your Android apps.

2. Build a basic app.

We'll build a simple app using Android Studio that will display some sample text on the screen.

3. Run the app in the Android emulator.

We'll use the built-in emulator to see the app up and running.

4. Change the app.

Finally, we'll make a few tweaks to the app we created in step 2, and run it again.

Your development environment

Java is the most popular language used to develop Android applications. Android devices don't run *.class* and *.jar* files. Instead, to improve speed and battery performance, Android devices use their own optimized formats for compiled code. That means that you can't use an ordinary Java development environment—you also need special tools to convert your compiled code into an Android format, to deploy them to an Android device and to let you debug the app once it's running.

All of these come as part of the **Android SDK**.

The Android SDK

The Android Software Development Kit contains the libraries and tools you need to develop Android apps:

SDK Platform
There's one of these for each version of Android.

SDK Tools
Tools for debugging and testing, plus other useful utilities. It also features a set of platform dependent tools.

Sample apps
If you want practical code examples to help you understand how to use some of the APIs, the sample apps might help you.

These are just some of the main points.

Documentation
So you can get to the latest API documentation offline.

Android support
Extra APIs that aren't available in the standard platform.

Google Play Billing
Allows you in integrate billing services in your app.

Install Android Studio

1. Create a new project

When you're done, you should see the Android Studio welcome screen. You're ready to build your first Android app.

The Android Studio welcome screen gives you a number of 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".

This is the Android Studio welcome screen. It includes a set of options for things you can do.

Build a basic app

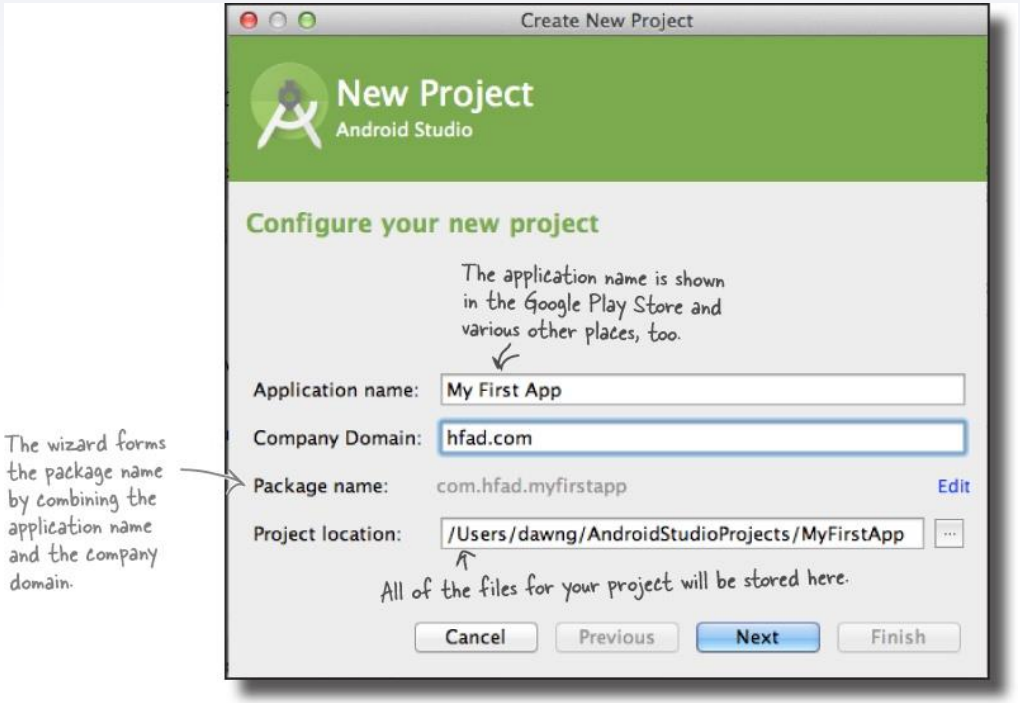
Now that you've set up your development environment, you're ready to create your first Android app. Here's what the app will look like:

You now need to configure the app by saying what you want to call it, what company domain to use, and where you would like to store the files.

Android Studio uses the company domain and application name to form the name of the package that will be used for your app.

As an example, if you give your app a name of "My First App" and use a company domain of "hfad.com", Android Studio will derive a package name of com.hfad.myfirstapp. The package name is really important in Android, as it's used by Android devices to uniquely identify your app.

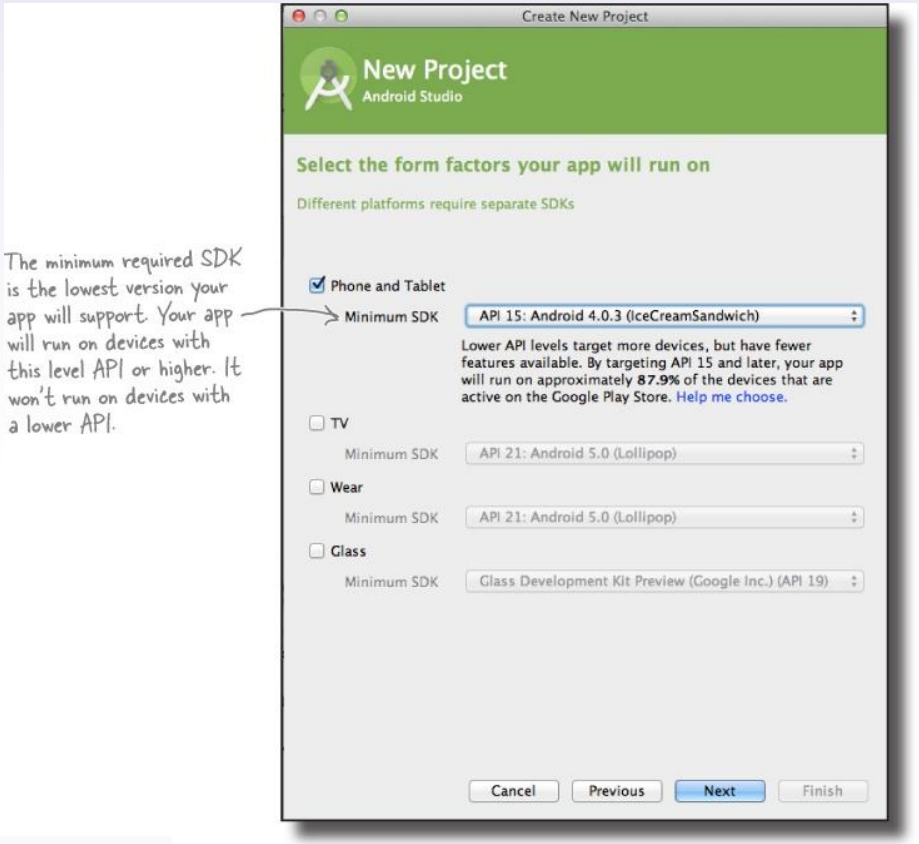
Enter an application name of "My First App", a company name of "hfad.com", and accept the default project location. Then click on the Next button.



3. Specify the API level

You now need to indicate which API level of Android your app will use. API levels increase with every new version of Android. Unless you only want your app to run on the very newest devices, you probably want to specify one of the older APIs.

When you've done this, click on the Next button.



What is API level?

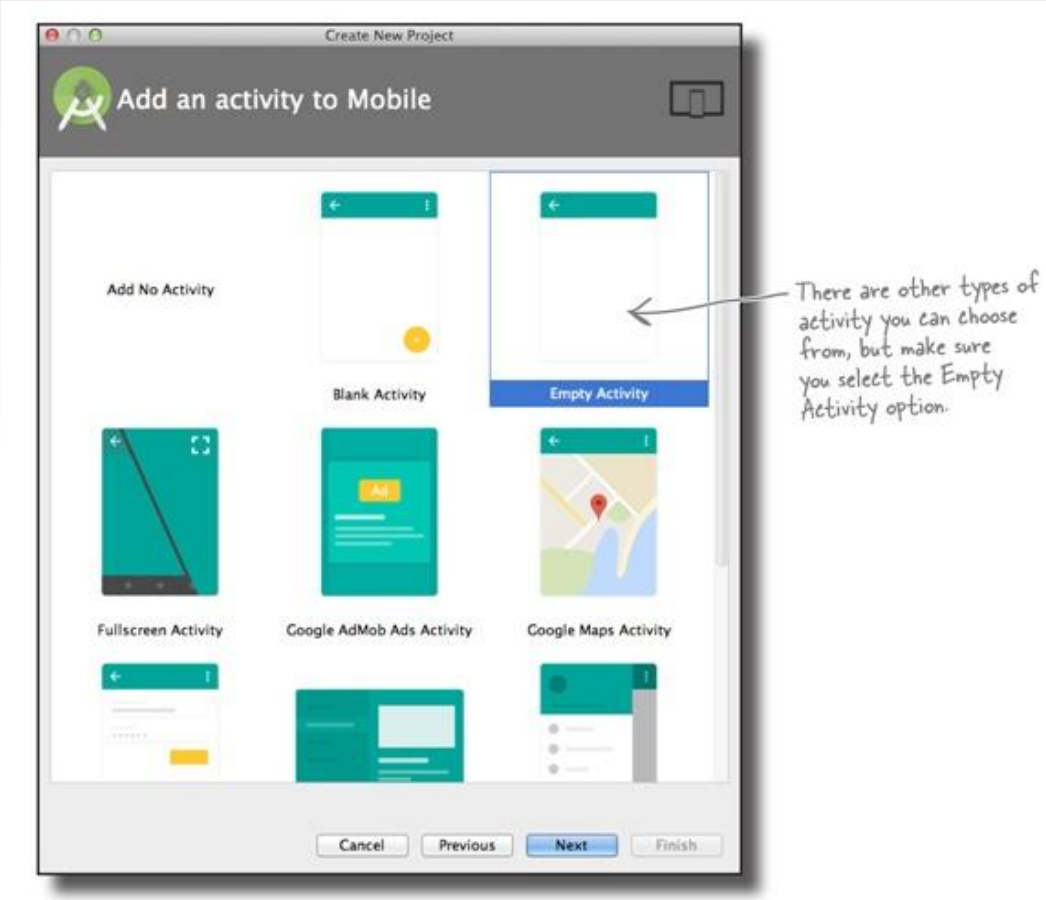
> API Level is an integer value that uniquely identifies the framework API revision offered by a version of the Android platform.

4. Create an activity

The next screen gives you a series of templates you can use to create an activity and layout.

You need to choose one.

We're going to create an app with a basic activity and layout, so choose the Empty Activity option and click the Next button.

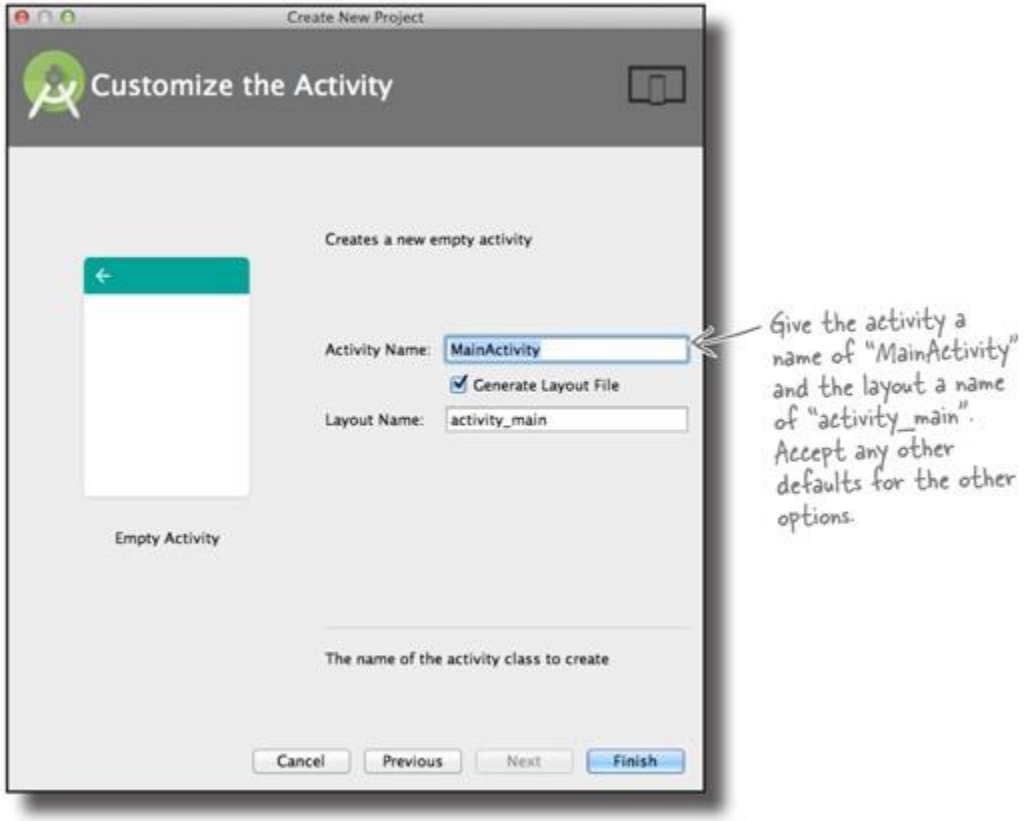


5. Configure the activity

You will now be asked what you want to call the screen's activity and layout. You will also need to say what the title of the screen will be, and specif a menu resource name. Enter an activity name of "MainActivity", and a layout name of "activity_main".

The activity is a Java class, and the layout is an XML file, so the names we've given here will crea a Java class file called *MainActivity.java* and an XML file called *activity_main.xml*.

When you click on the Finish button, Android Studio will build your app.



YOU'VE JUST CREATED YOUR FIRST ANDROID APP

So what just happened?

- The Android Studio wizard created a project for your app, configured to your specifications.

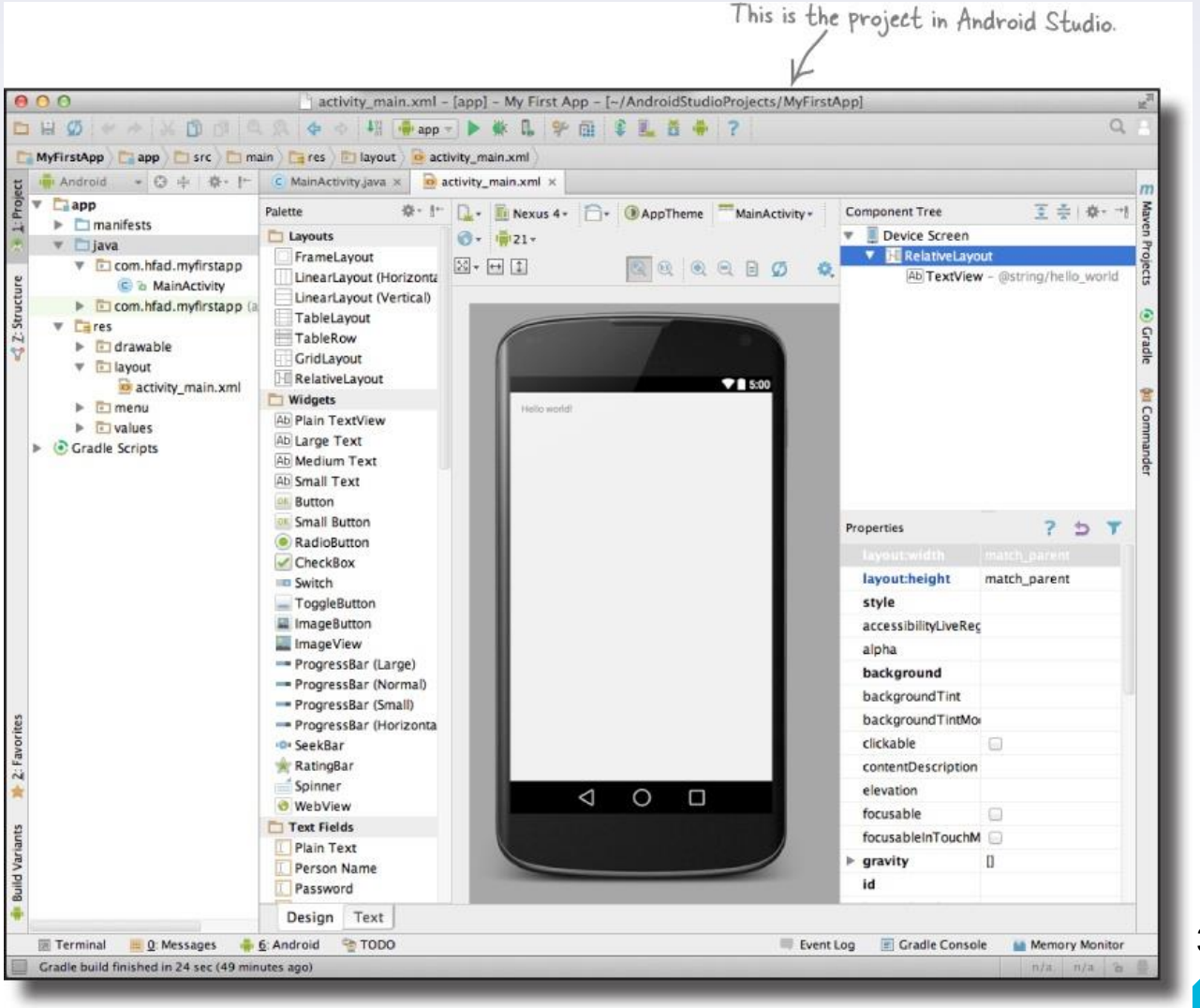
You defined which versions of Android the app should be compatible with, and the wizard created all of the files and folders needed for a basic valid app.

- It created a basic activity and layout with template code.

The template code includes layout XML and activity Java code, with sample “Hello world!” text in the layout. You can change this code.

When you finish creating your project by going through the wizard, Android Studio automatically displays the project for you.

Here's what our project looks like



Android Studio creates a complete folder structure for you

An Android app is really just a bunch of valid files in a particular folder structure, and Android Studio sets all of this up for you when you create a new app.

The easiest way of looking at this folder structure is with the explorer in the leftmost column of Android Studio.

The explorer contains all of the projects that you currently have open.

To expand or collapse folders, just click on the arrows to the left of the folder icons.

The folder structure includes different types of files

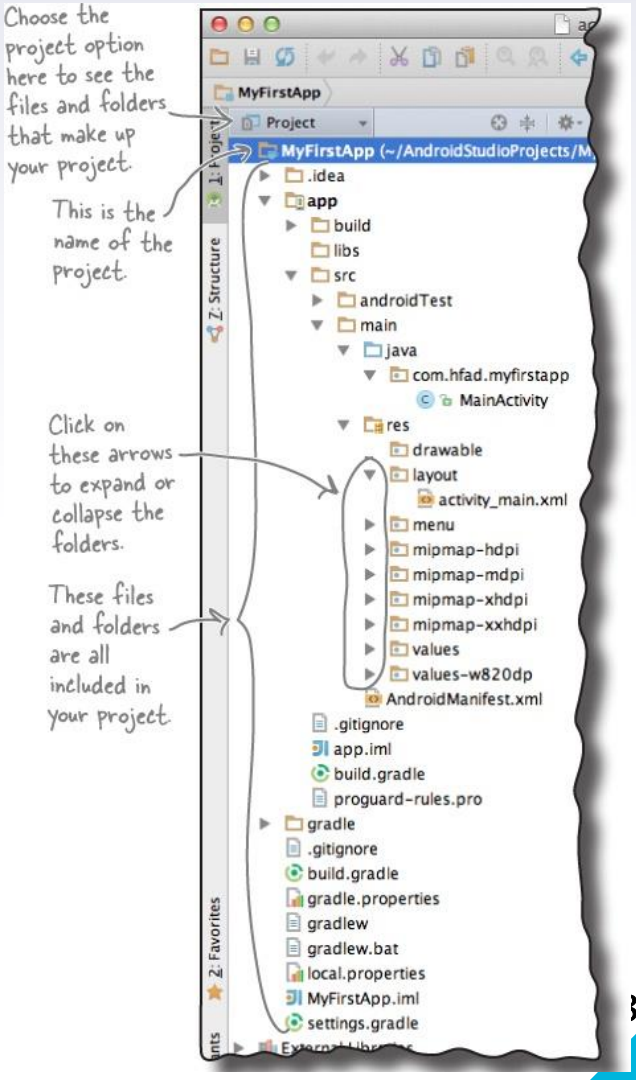
If you browse through the folder structure, you'll see that the wizard has created various types of files and folders for you:

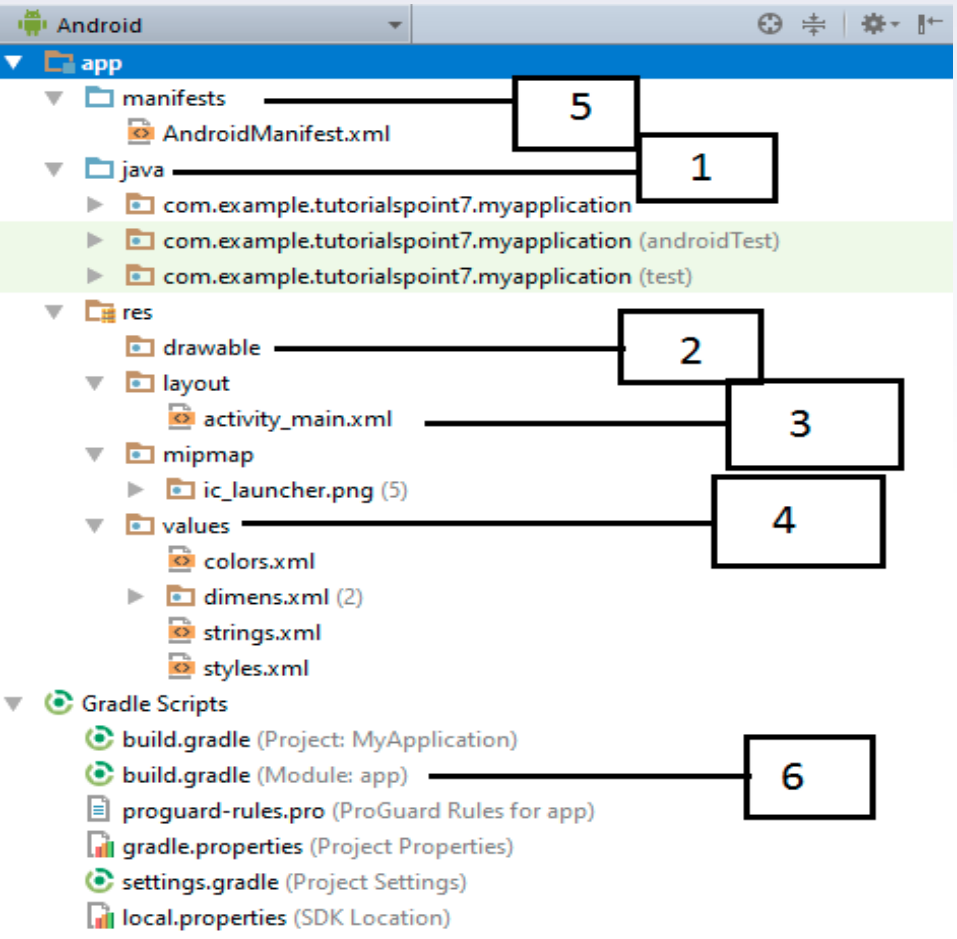
Java and XML source files: These are the activity and layout files the wizard created for you.

Android-generated Java files: There are some extra Java files you don't need to touch which Android Studio generates for you automatically.

Resource files: These include default image files for icons, styles your app might use, and any common String values your app might want to look up.

Android libraries: In the wizard, you specified the minimum SDK version you want your app to be compatible with. Android Studio makes sure it includes the relevant Android libraries for this version.

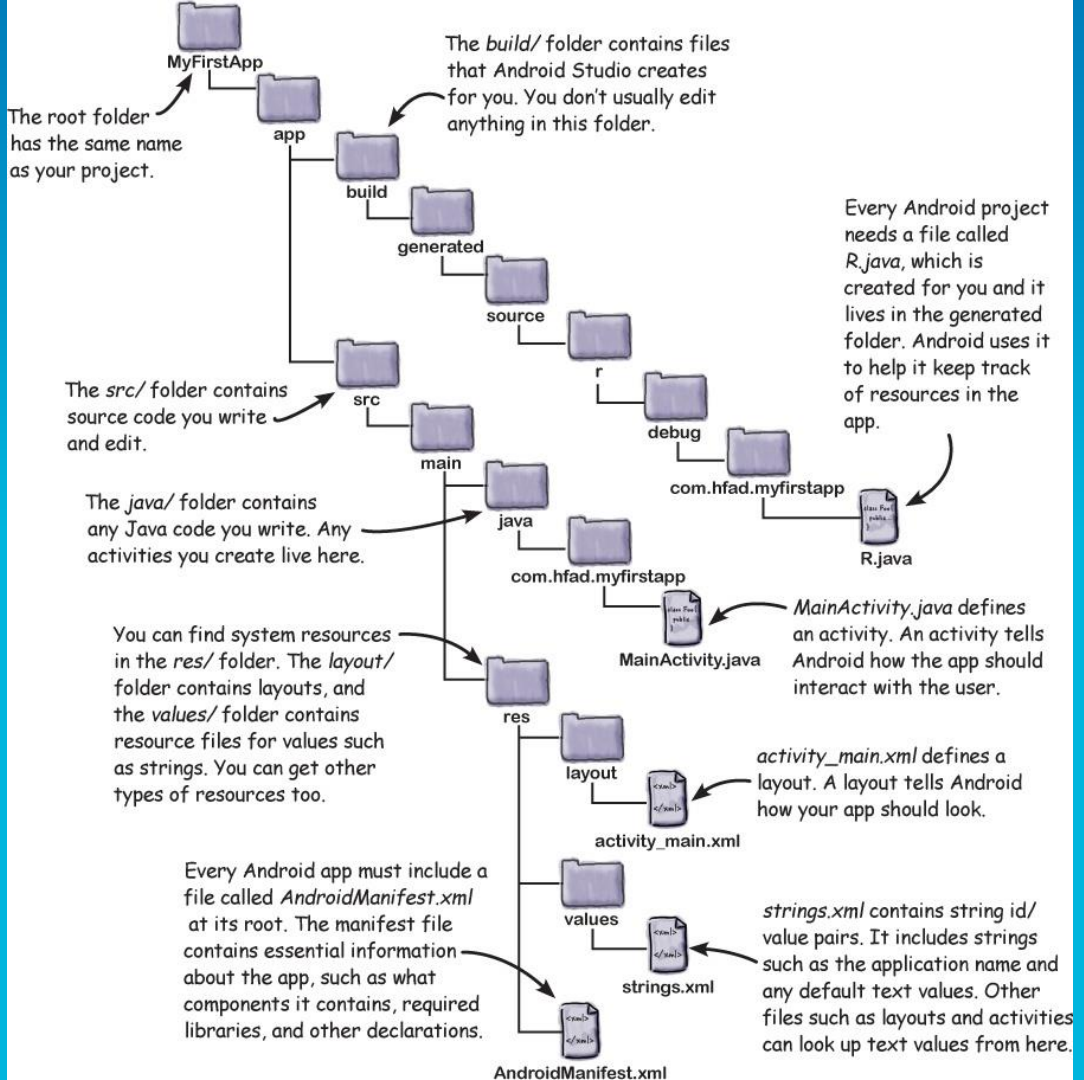




Anatomy of Android Application

Sr.No.	Folder, File & Description
1	**Java** This contains the .java source files for your project. By default, it includes an MainActivity.java source file having an activity class that runs when your app is launched using the app icon.
2	**res/drawable-hdpi** This is a directory for drawable objects that are designed for high-density screens.
3	**res/layout** This is a directory for files that define your app's user interface.
4	**res/values** This is a directory for other various XML files that contain a collection of resources, such as strings and colours definitions.
5	**AndroidManifest.xml** This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.
6	**Build.gradle** This is an auto generated file which contains compileSdkVersion, buildToolsVersion, applicationId, minSdkVersion, targetSdkVersion, versionCode and versionName

Useful files in your project



The Main Activity File

The main activity code is a Java file MainActivity.java. This is the actual application file which ultimately gets converted to a Dalvik executable and runs your application.

```
package com.example.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
}
```

Here, *R.layout.activity_main* refers to the *activity_main.xml* file located in the *res/layout* folder. The *onCreate()* method is one of many methods that are figured when an activity is loaded.

The Manifest File

Whatever component you develop as a part of your application, you must declare all its components in a *manifest.xml* which resides at the root of the application project directory.

This file works as an interface between Android OS and your application, so if you do not declare your component in this file, then it will not be considered by the OS.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.tutorialspoint7.myapplication">

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">

    <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>
```

The Strings File

The strings.xml file is located in the *res/values* folder and it contains all the text that your application uses. For example, the names of buttons, labels, default text, and similar types of strings go into this file. This file is responsible for their textual content.

strings.xml is the default resource file used to hold name/value pairs of strings so that they can be referenced throughout your app. It has the following format:

android:text means that this is the text property of the <TextView> element, so it specifies which text should be displayed in the layout. But why does it say "@string/hello world" rather than “Hello world!”? What does this actually mean?

Let’s start with the first part, @string. This is just a way of telling Android to look up a text value from a string resource file. In our case, Android Studio created a string resource file for us called strings.xml, located in the app/src/main/res/values folder.

Put string values in strings.xml rather than hardcoding them. strings. xml is a resource file used to hold name/value pairs of strings. Layouts and activities can look up string values using their name.

The second part, hello_world, tells Android to look up the value of a resource with the name hello_world. So @string/hello_world means “look up the string resource with the name hello_world, and use the associated text value.”

There's one key reason: localization

Say you've created an app and it's a big hit on your local Google Play Store. But you don't want to limit yourself to just one country or language—you want to make it available internationally and for different languages.

Separating out text values into *strings.xml* makes dealing with issues like this much easier. Rather than having to change hardcoded text values in a whole host of different activity and layout files, you can simply replace the *strings.xml* file with an internationalized version.

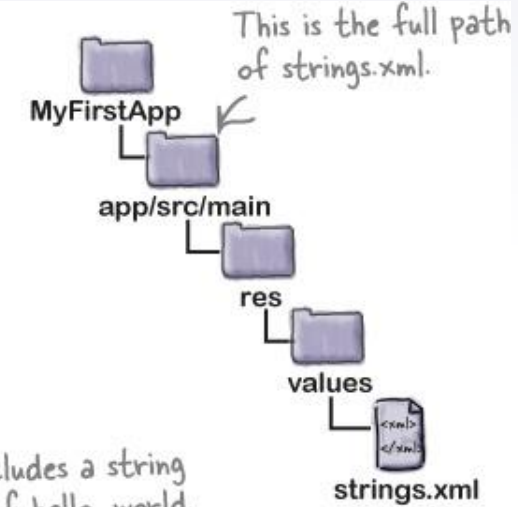
Using *strings.xml* as a central resource for text values also makes it easier to make global changes to text across your whole application. If your boss needs you to change the wording in an app because the company's changed its name, only *strings.xml* needs to be changed.

Let's look in the strings.xml file

Android Studio created a string resource file for us called strings.xml, so let's see if it contains a hello_world resource. Use the explorer to find it in the app/src/main/res/values folder, and open it by double-clicking on it

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My First App</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
</resources>
```

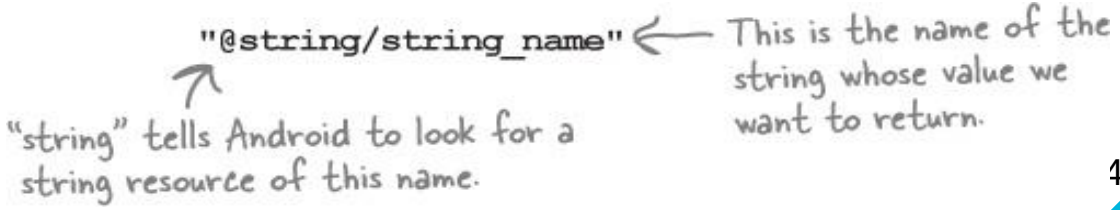
strings.xml includes a string with a name of hello_world, and a value of "Hello world!".



As you can see, there's a line of code that looks just like what we are looking for. It describes a string resource with a name of hello_world, and a value of "Hello world!":Once you've updated the file, go to the File menu and choose the Save All option to save your change.

- There are two things that allow Android to recognize strings.xml as being a string resource file:
- The file is held in the folder app/src/main/res/values.
- XML files held in this folder contain simple values, such as strings and colors.
- The file has a <resources> element, which contains one or more <string> elements.
- The format of the file itself indicates that it's a resource file containing Strings. The <resources> element tells Android that the file contains resources, and the <string> element identifies each String resource.
- This means that you don't need to call your String resource file strings.xml; if you want, you can call it something else, or split your Strings into multiple files.
- Each name/value pair takes the form <string name:"button_one">London Bridge</string> where string_name is the identifier of the string, and string_value is the String value itself.

A layout can retrieve the value of the String using



Update strings.xml to change the text

- So let's change the sample text in the app. If you've not already done so, find the file *strings.xml* in the Android Studio explorer, and double-click on it to open it.
- Here's the code from the file. You need to look for the string with the name "hello_world", and change its corresponding text value from "Hello world!" to "Sup doge":

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My First App</string>
    <string name="hello_world">Hello world! Sup doge</string>
    <string name="action_settings">Settings</string>
</resources>
```

Change the value here from "Hello world!" to "Sup doge".

Android R.java file

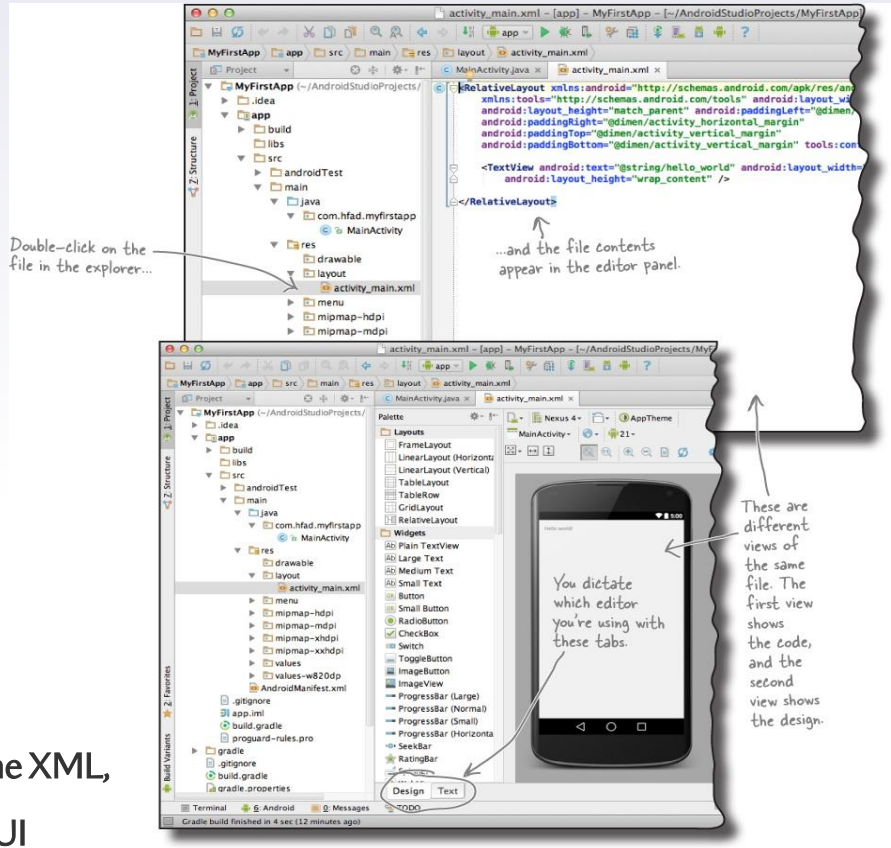
Android R.java is an auto-generated file by aapt (Android Asset Packaging Tool) that contains resource IDs for all the resources of res/ directory.

If you create any component in the activity_main.xml file, id for the corresponding component is automatically created in this file. This id can be used in the activity source file to perform any action on the component.

Note: If you delete R.jar file, android creates it automatically.

Edit code with the Android Studio editors

You view and edit files using the Android Studio editors. Double-click on the file you want to work with, and the file contents will appear in the middle of the Android Studio window.



The code editor

Most files get displayed in the code editor. The code editor is just like a text editor, but with extra features such as color coding and code checking.

The design editor

If you're editing a layout, you have an extra option. Rather than edit the XML, you can use the design editor. The design editor allows you to drag GUI components onto your layout, and arrange them how you want. The code editor and design editor give different views of the same file, so you can switch back and forth between the two.

45

activity_main.xml

```
<RelativeLayout 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:paddingLeft="16dp"
android:paddingRight="16dp"
android:paddingTop="16dp"
android:paddingBottom="16dp"
tools:context=".MainActivity">

<TextView
    android:text="@string/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

</RelativeLayout>
```

Add padding to the screen margins.

Display the value of a string resource called `hello_world`.

Include a `TextView` GUI component for displaying text.

Make the layout the same width and height as the screen size on the device.

Make the text wrap horizontally and vertically.

MainActivity.java

```
package com.hfad.myfirstapp;

import android.os.Bundle;
import android.app.Activity;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

This is the package name.

These are Android classes used in `MainActivity`.

Specifies which layout to use.

Implement the `onCreate()` method from the `Activity` class. This method is called when the activity is first created.

`MainActivity` extends the Android class `android.app.Activity`.

You have a couple of options when it comes to running your apps.

The first option is to run them on a physical device.

But what if you don't have one with you, or you want to see how it looks on a type of device you don't have?

An alternative option is to use the Android emulator that's built into the Android SDK.

The emulator enables you to set up one or more Android virtual devices (AVDs) and then run your app in the emulator as though it's running on a physical device.

The Android emulator allows you to run your app on an Android virtual device (AVD). The AVD behaves just like a physical Android device. You can set up numerous AVDs, each emulating a different type of device.

What does the emulator look like?

- The emulator is an application that re-creates the exact hardware environment of an Android device: from its CPU and memory, through to the sound chips and the video display. The emulator is built on an existing emulator called QEMU, which is similar to other virtual machine applications you may have used, like VirtualBox or VMWare.

- The exact appearance and behavior of the AVD depends on how you've set up the AVD in the first place.

Once you've set up an AVD, you can see your app running on it. Android Studio launches the emulator for you.

Just like a physical phone, you need to unlock it before you start using it. Simply click on the lock icon and drag it upward.

Creating an Android Virtual Device

- There are a few steps you need to go through in order to set up an AVD within Android Studio. We'll set up a Nexus 4 AVD running API level 21 so that you can see how your app looks and behaves running on this type of device. The steps are pretty much identical no matter what type of device you want to set up.

Open the Android Virtual Device Manager

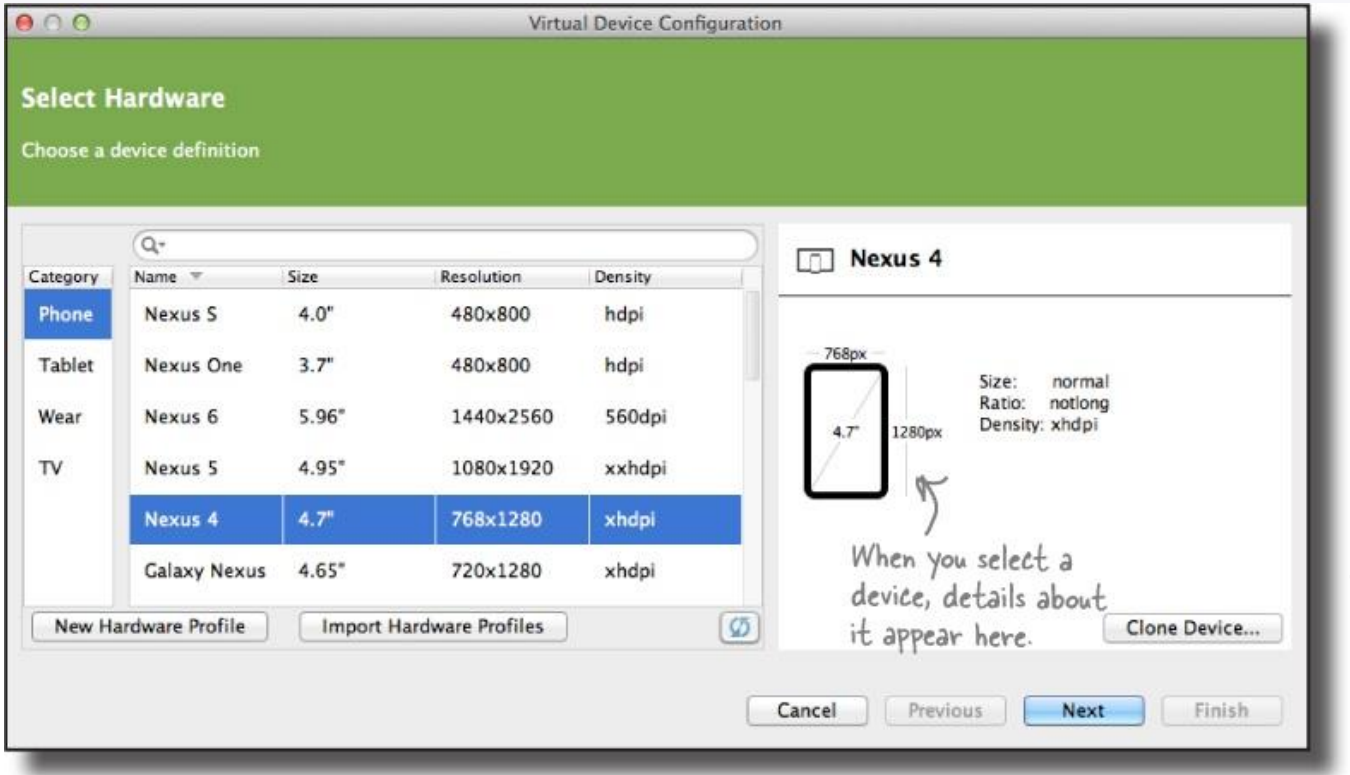
Click on the "Create a virtual device" button to create an AVD.

The AVD Manager allows you to set up new AVDs, and view and edit ones you've already created. Open it by selecting Android on the Tools menu and choosing AVD Manager.

If you have no AVDs set up already, you'll be presented with a screen prompting you to create one. Click on the "Create a virtual device" button.

Select the hardware

On the next screen, you'll be prompted to choose a device definition. This is the type of device your AVD will emulate. You can choose a variety of phone, tablet, wear, or TV devices.



We're going to see what our app looks like running on a Nexus 4 phone.

Choose Phone from the Category menu and Nexus 4 from the list. Then click the Next button.

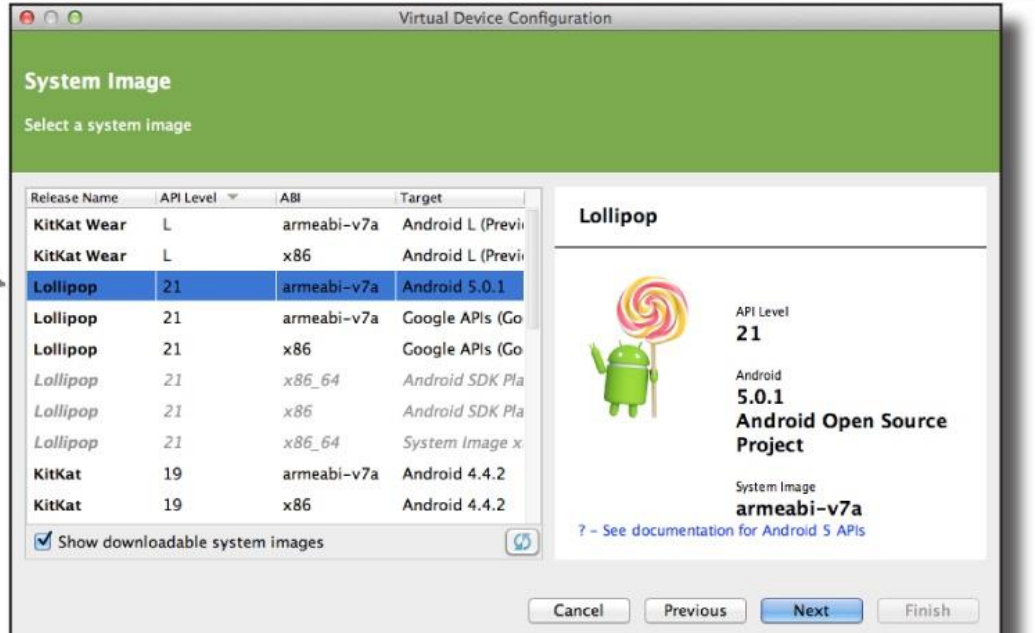
Select a system image

Next, you need to select a system image. The system image gives you an installed version of the Android operating system. You can choose the version of Android you want to be on your AVD, and what type of CPU (ARM or x86).

You need to choose a system image for an API level that's compatible with the app you're building.

Then click on the Next button.

If you don't have this system image installed, you'll be given the option to download it.



As an example, if you want your app to work on a minimum of API level 15, choose a system image for *at least* API level 15. We're going to use a system image for API level 21.

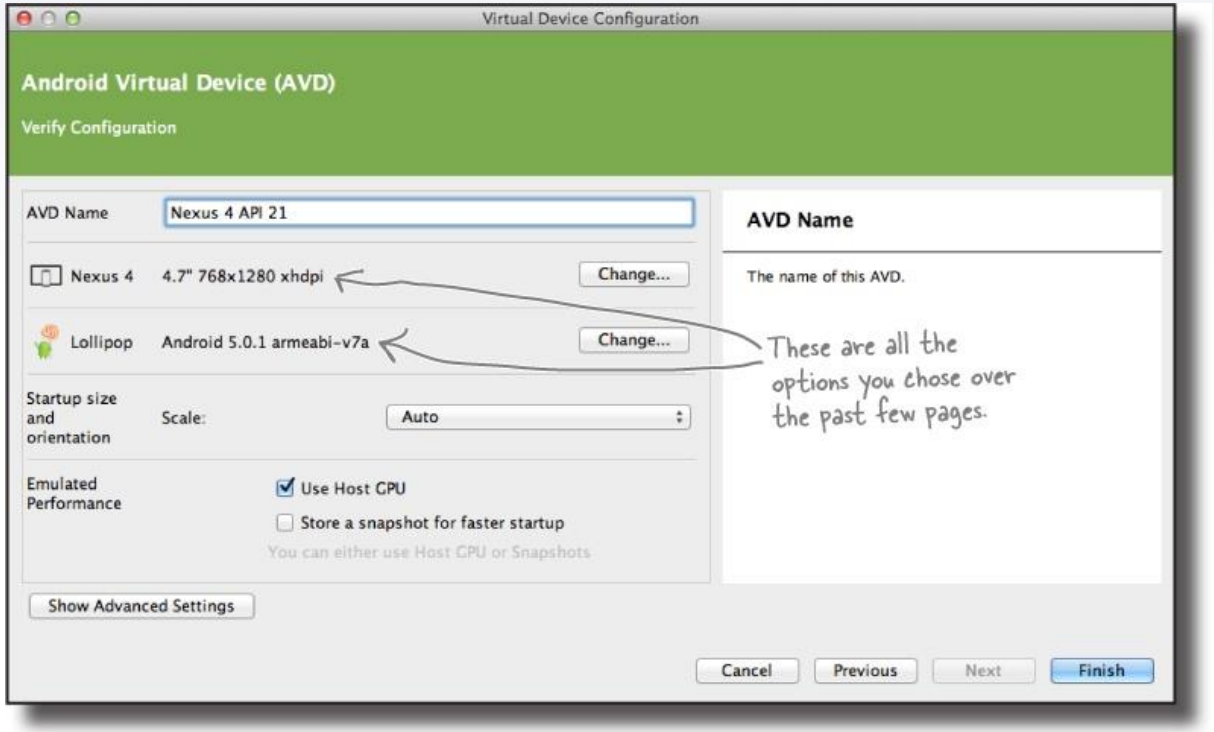
Choose the option for Lollipop 21 armeabi-v7a with a target of Android 5.0.1.

Verify the AVD configuration

On the next screen, you'll be asked to verify the AVD configuration.

This screen summarizes the options you chose over the last few screens, and gives you the option of changing them.

Accept the options, and click on the Finish button.



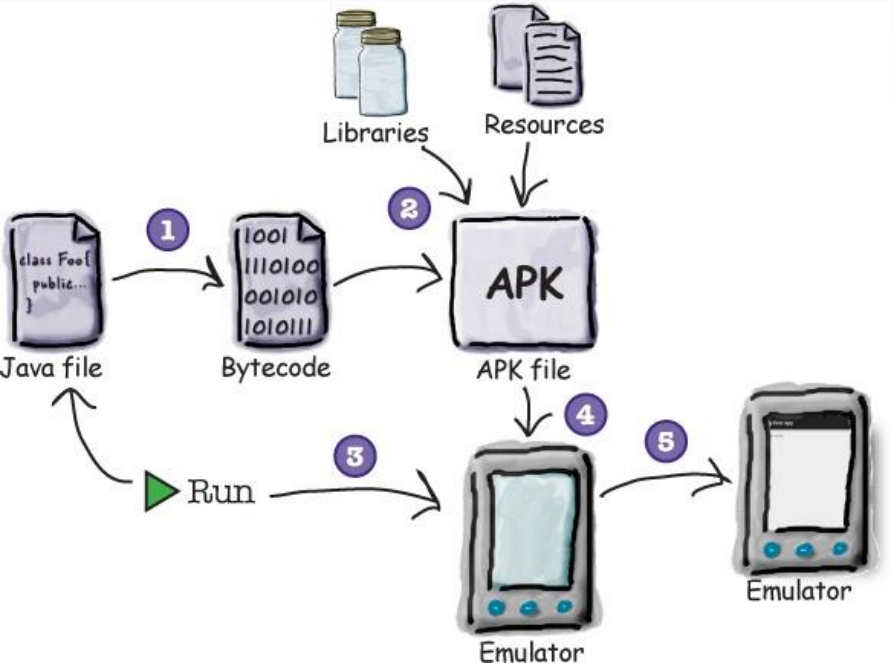
The AVD Manager will create the AVD for you, and when it's done, display it in the AVD Manager list of devices. You may now close the AVD Manager.

Now that you've set up your AVD, let's run the app on it. To do this, choose the "Run 'app'" command from the Run menu. When you're asked to choose a device, make sure the "Launch emulator" option is selected, along with the Nexus 4 AVD you just created. Then click on the OK button.

Compile, package, deploy and run

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:

An APK file is an Android application package. It's basically a JAR or ZIP file for Android applications.



1.The Java source files get compiled to bytecode.

2.An Android application package, or APK file, gets created.

3.The APK file includes the compiled Java files, along with any libraries and resources needed by your app.

4.Assuming there's not one already running, the emulator gets launched with the AVD.

5.Once the emulator has been launched and the AVD is active, the APK file is uploaded to the AVD and installed.

6.The AVD starts the main activity associated with the app.

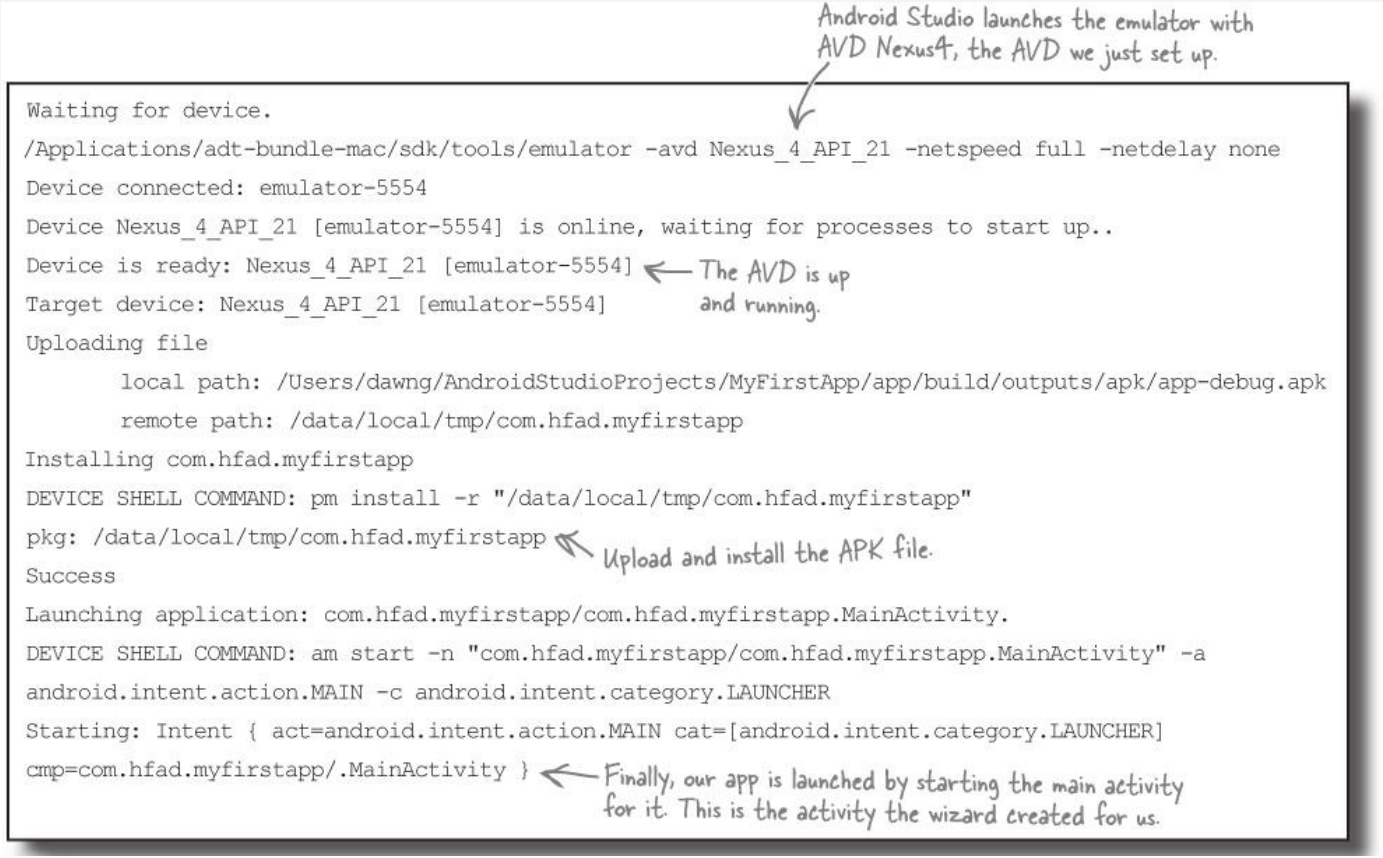
7.Your app gets displayed on the AVD screen, and it's all ready for you to test out.

You can watch progress in the console

- It can sometimes take quite a while for the emulator to launch with your AVD—often *several minutes*. The great news is that you can see what's happening using the Android Studio console. The console gives you a blow-by-blow account of what the gradle build system is doing, and if it encounters any errors, you'll see them highlighted in the text.

You can find the console at the bottom of the Android Studio screen:

Here's the output from our console window when we ran our app:



So let's look at what actually happens on screen when you run your app.

First, the emulator fires up in a separate window.

The emulator takes a while to load the AVD, but then after a bit you see the locked screen of the AVD.

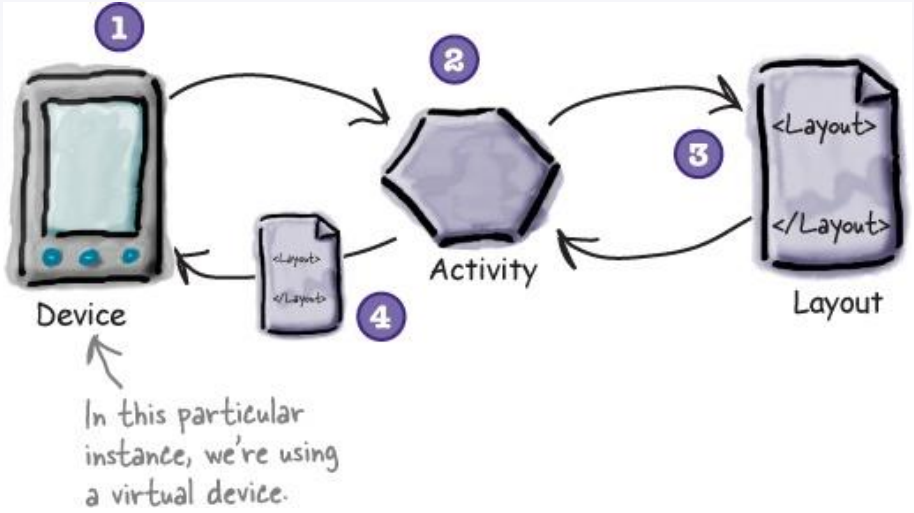
When you unlock the AVD screen by swiping the padlock icon upward, you see the app you just created.

The application name appears at the top of the screen, and the default sample text "Hello world!" is displayed in the screen.

What just happened?

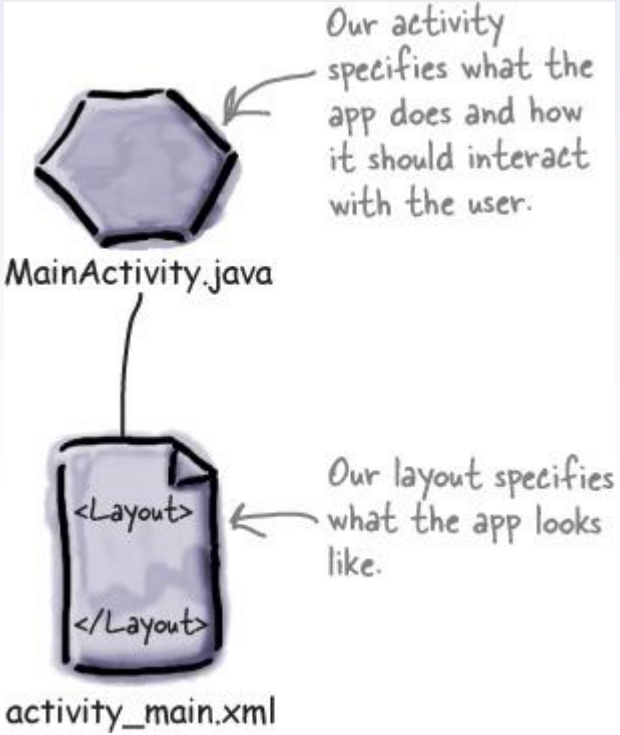
Let's break down what happens when you run the app:

1. Android Studio launches the emulator, loads the AVD, and installs the app.
2. When the app gets launched, an activity object is created from MainActivity.java.
3. The activity specifies that it uses the layout activity_main.xml.
4. The activity tells Android to display the layout on the screen. The text "Hello world!" gets displayed.



Activities and layouts

- An activity is a single, defined thing that your user can do.
- Activities are usually associated with one screen, and they're written in Java.
- 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.

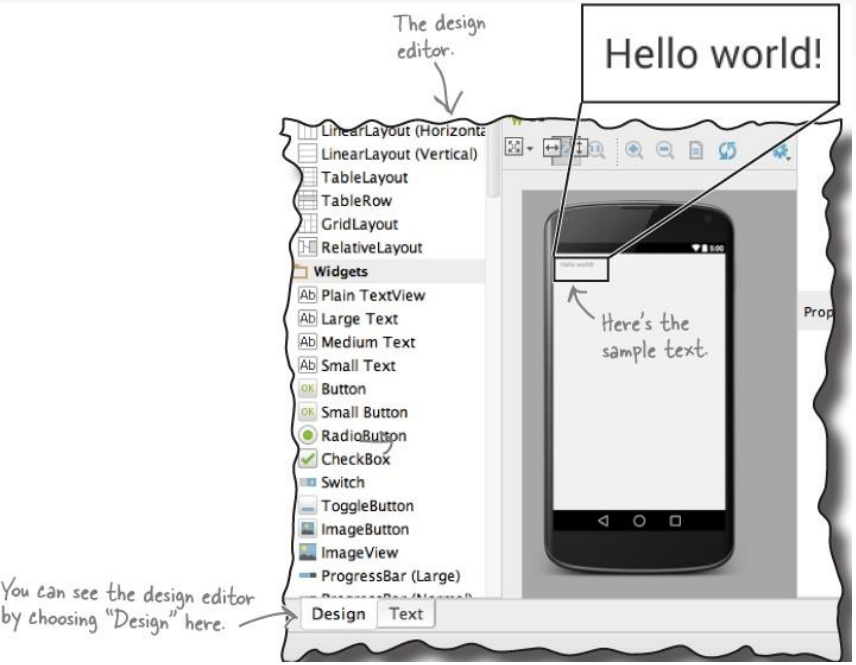


Layouts define how the user interface is presented.Activities define actions.

We need to change the sample "Hello world!" text that Android Studio created for us, so let's start with the layout file *activity_main.xml*. If it isn't already open in an editor, open it now by finding the file in the *app/src/main/res/layout* folder in the explorer and double-clicking on it.

The design editor

There are two ways of viewing and editing layout files in Android Studio: through the design editor and through the code editor.



The code editor

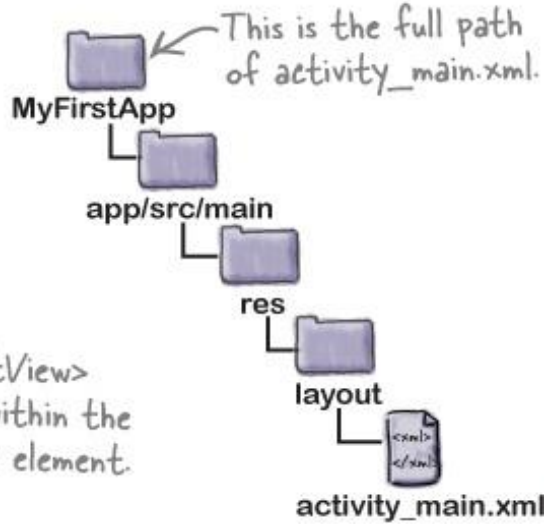
When you choose the code editor option, the content of *activity_main.xml* is displayed. Let's take a closer look at it.

```
<RelativeLayout 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"
    ...
    tools:context=".MainActivity" >

    <TextView
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

This is the <RelativeLayout> element.

Android Studio gave us more XML here, but you don't need to think about that just yet.

This is the <TextView> element nested within the <RelativeLayout> element.



The code contains two elements.

The first element is the <RelativeLayout> element. This element tells Android to display items on the layout in relative positions. You can use <RelativeLayout>, for instance, to center items in the middle of the layout, align them to the bottom of the screen on your Android device, or position them relative to other items.

The second element is the <TextView> element. This element is used to display text to the user. It's nested within the <RelativeLayout>, and in our case it's being used to display the sample text "Hello world!".

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.

dip, dp, sip, sp & px

Pixels(px) – corresponds to actual pixels on the screen. This is used if you want to give in terms of absolute pixels for width or height.

Sp or sip - Scaleable Pixels OR scale-independent pixels- this is like the dp unit, but it is also scaled by the user's font size preference. It is recommend you use this unit when specifying font sizes, so they will be adjusted for both the screen density and user's preference.

dp or dip - Density-independent Pixels - an abstract unit that is based on the physical density of the screen. These units are relative to a 160 dpi screen, so one dp is one pixel on a 160 dpi screen. The ratio of dp-to-pixel will change with the screen density, but not necessarily in direct proportion. Note: The compiler accepts both "dip" and "dp", though "dp" is more consistent with "sp".

Unit	Description	Units Per Physical Inch	Density Independent	Same Physical Size On Every Screen
px	Pixels	Varies	No	No
in	Inches	1	Yes	Yes
mm	Millimeters	25.4	Yes	Yes
pt	Points	72	Yes	Yes
dp	Density Independent Pixels	~160	Yes	No
sp	Scale Independent Pixels	~160	Yes	No

Views

All the interaction of a user with the Android application is through the user interface(UI), hence it is very important to understand the basics about the User Interface of an android application.

View is the basic building block of UI(User Interface) in android.

View refers to the android.view.View class, which is the super class for all the GUI components like TextView, ImageView, Button etc.

View can be considered as a rectangle on the screen that shows some type of content. It can be an image, a piece of text, a button or anything that an android application can display. The rectangle here is actually invisible, but every view occupies a rectangle shape.

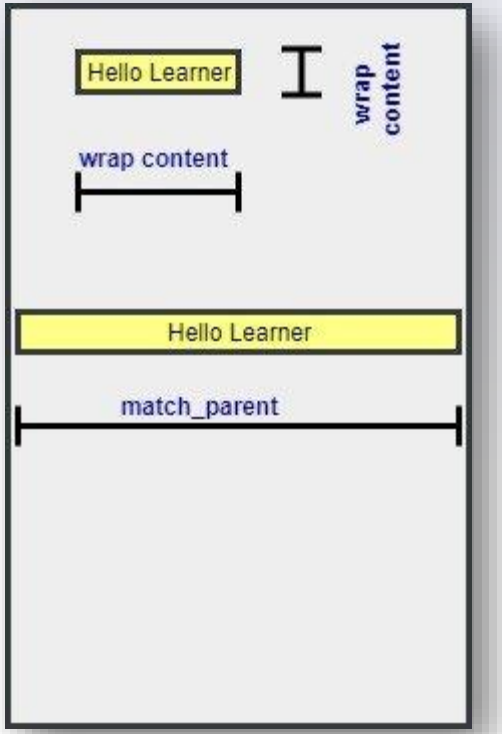
The size of the view can set it manually, by specifying the exact size(with proper units) or by using some predefined values.

These predefined values are match_parent and wrap_content.

match_parent means it will occupy the complete space available on the display of the device.

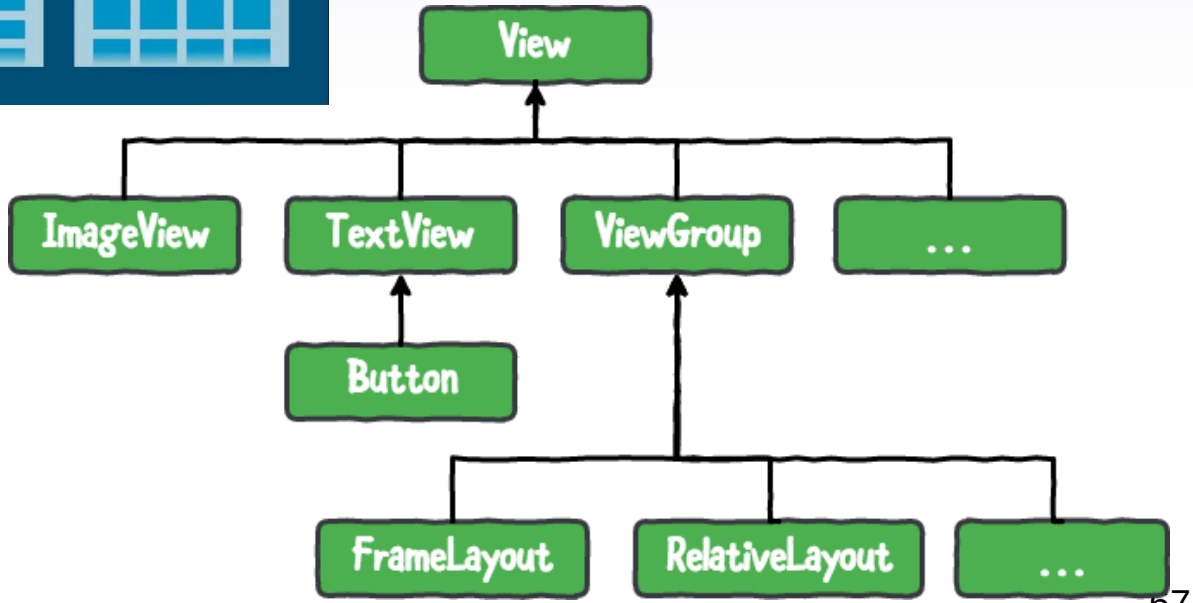
wrap_content means it will occupy only that much space as required for its content to display.

A View is also known as Widget in Android. Any visual(that we can see on screen) and interactive(with which user can interact with) is called a Widget.

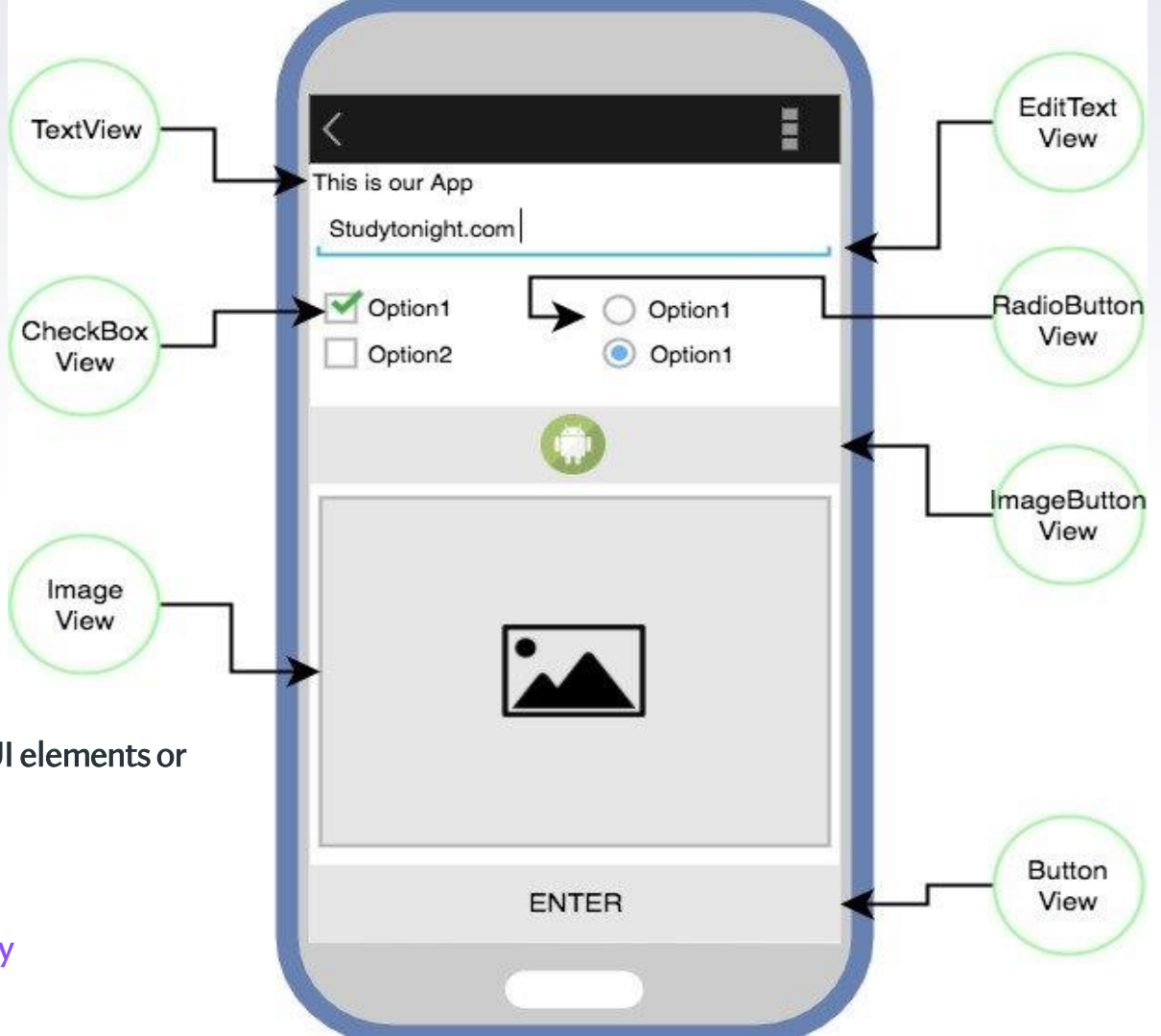


What Is A ViewGroup?

A ViewGroup is a special view that can contain other views (called children.) The view group is the base class for layouts and views containers.



Most commonly used Android View classes



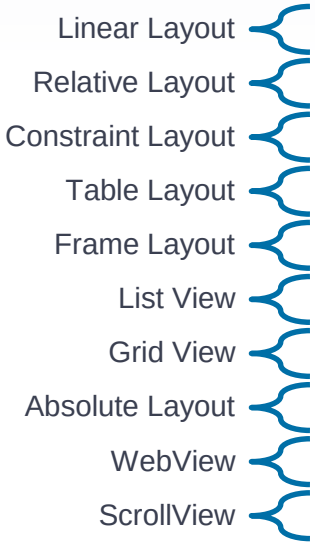
The Android framework will allow us to use UI elements or widgets in two ways:

- Use UI elements in the XML file
- Create elements in the Kotlin file dynamically

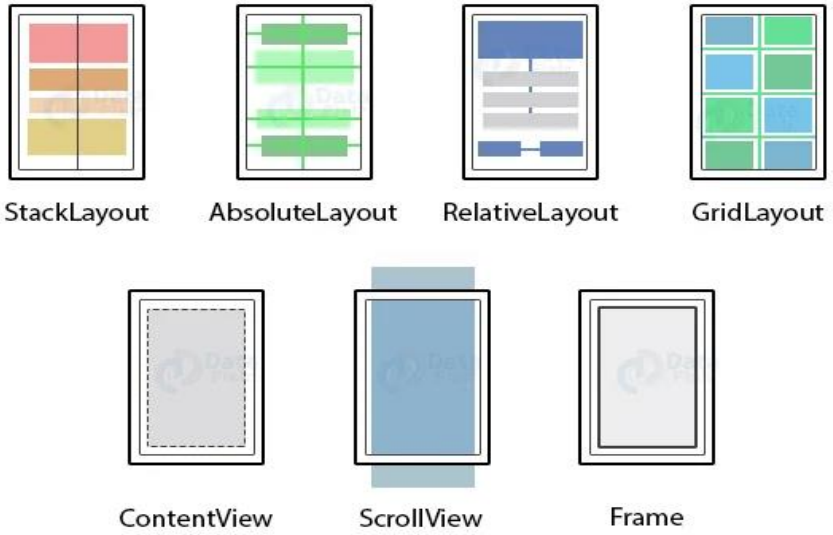
What is Android Layout?

Layout basically refers to the arrangement of elements on a page these elements are likely to be images, texts or styles.

All elements in the layout are built with the help of Views and ViewGroups. These layouts can have various widgets like buttons, labels, textboxes, and many others.



Types of Android Layouts



Android Linear Layout: LinearLayout is a ViewGroup subclass, used to provide child View elements one by one either in a particular direction either horizontally or vertically based on the orientation property.

Android Relative Layout: RelativeLayout is a ViewGroup subclass, used to specify the position of child View elements relative to each other like (A to the right of B) or relative to the parent (fix to the top of the parent).

Android Constraint Layout: ConstraintLayout is a ViewGroup subclass, used to specify the position of layout constraints for every child View relative to other views present. A ConstraintLayout is similar to a RelativeLayout, but having more power.

Android Table Layout: TableLayout is a ViewGroup subclass, used to display the child View elements in rows and columns.

Android 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.

Android Web View: WebView is a browser that is used to display the web pages in our activity layout.

Android ListView: ListView is a ViewGroup, used to display scrollable lists of items in a single column.

Android Grid View: GridView is a ViewGroup that is used to display a scrollable list of items in a grid view of rows and columns.

The Layout File

The activity_main.xml is a layout file available in *res/layout* directory, that is referenced by your application when building its interface. You will modify this file very frequently to change the layout of your application.

Layout Type	Description
Linear Layout	Linear Layout is used to align your items in a linear fashion, either vertical or horizontal.
Relative Layout	Relative Layout is used when you wish to place views relative to each other's position.
Table Layout	Table Layout is used to display the items(such as text, image, etc) in the form of rows and columns.
Absolute Layout	Absolute Layout is used when you want to fix the position of an element in the screen.
Frame Layout	Frame Layout is used when you deal with fragments. This layout allows you only to have one child view.
List View	List View is used when you wish to display your items in the form of a list.
Grid View	Grid View is used when you wish to show your data in the form of a grid of one's, two's, three's, or so on.
Web View	Web View is a layout that is used mainly when dealing with web pages inside the application.
Scroll View	Scroll View allows your content to scroll vertically or horizontally. The point to note is that ScrollView has only one child view.
Constraint Layout	Constraint layout is the most adaptive layout among all the above. It allows you to put vertical, horizontal, top, and bottom constraints for any element.

71

Android UI Controls

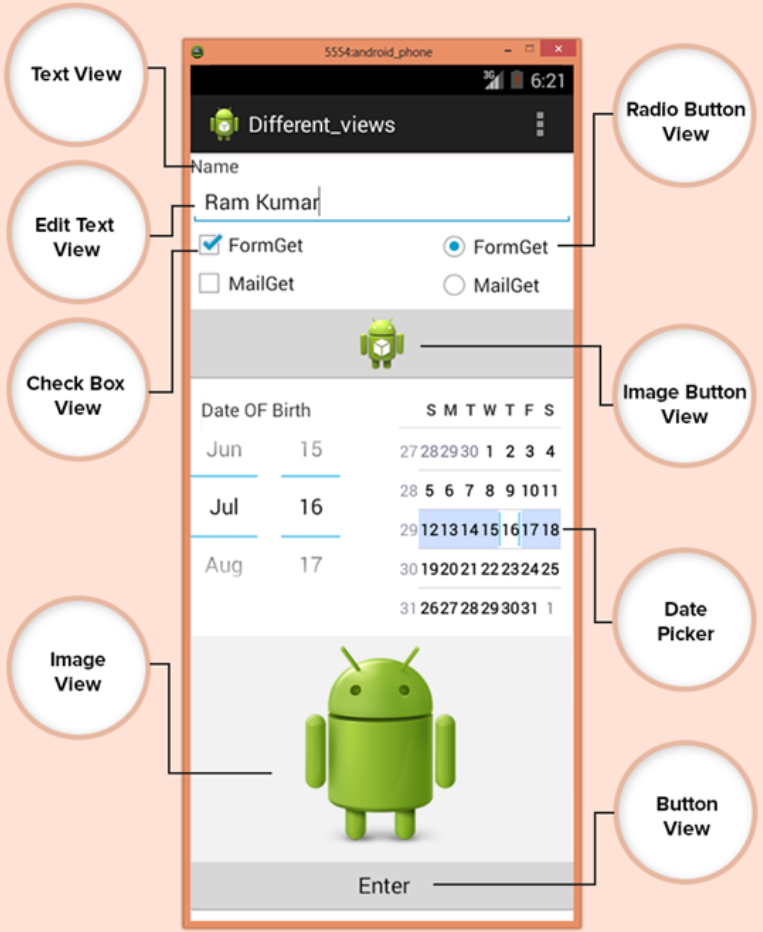
- ✓ Android UI Controls are those components of Android that are used to design the UI in a more interactive way.
- ✓ It helps us to develop an application that makes user interaction better with the view components. Android provides us a huge range of UI controls of many types such as buttons, text views, etc.
- ✓ UI is the only thing that a user interacts with within an Application. This is the reason that we make our Application look aesthetic and, more and more connective.
- ✓ To do so, we need to add the UI controls or we say Input controls in the respective application.

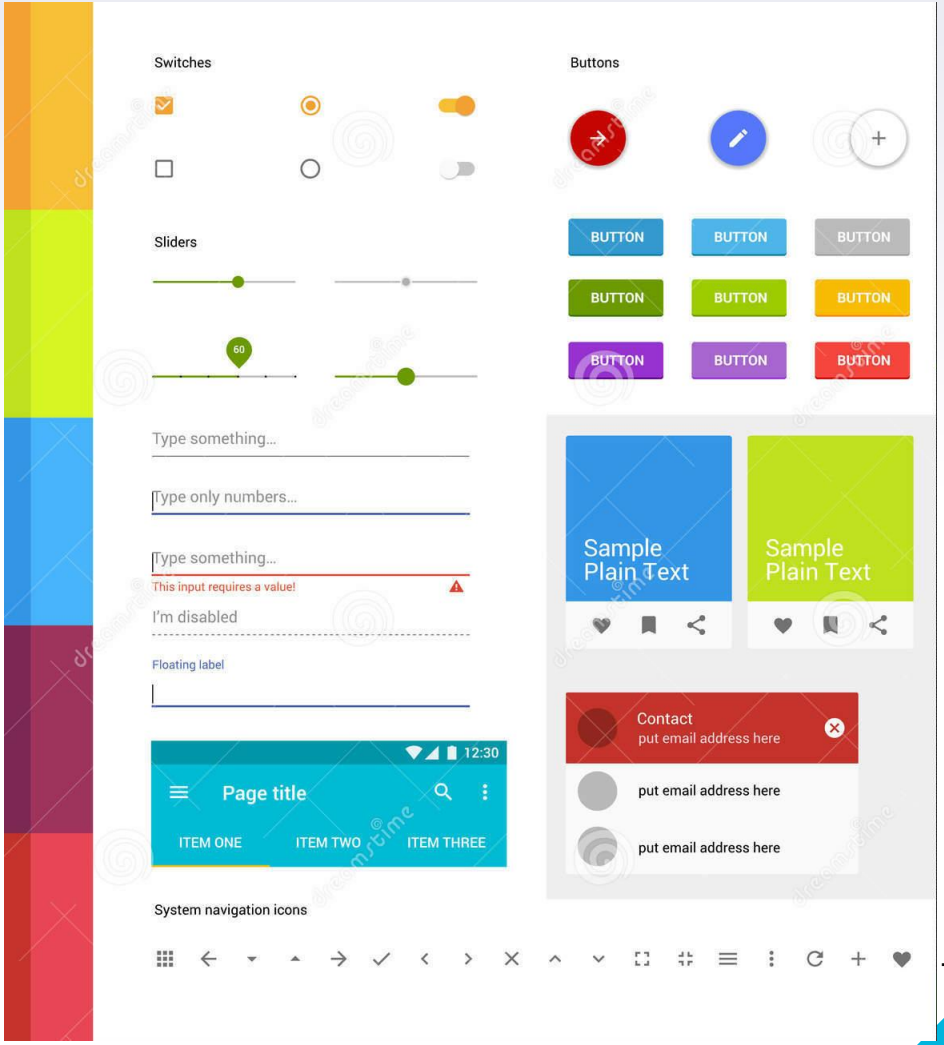
Each UI element has its specific attributes.

Some of the common attributes among the elements are as follows:
android:id: It is used to identify your UI element in your project uniquely.
android:height: It is used to set the vertical size of your element.
android:width: It is used to set the width of your element.
android:margin: It is used to provide margin around the elements.
android:padding: It is used to add padding around the content of the component.

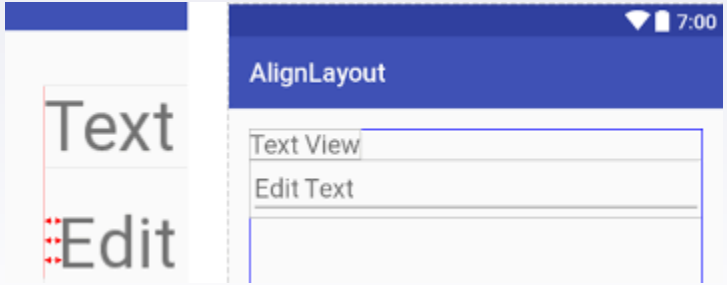
There are many more attributes that are specific to specif elements. There you will get to know while we proceed in the course.

Different Views In Android





Android TextView



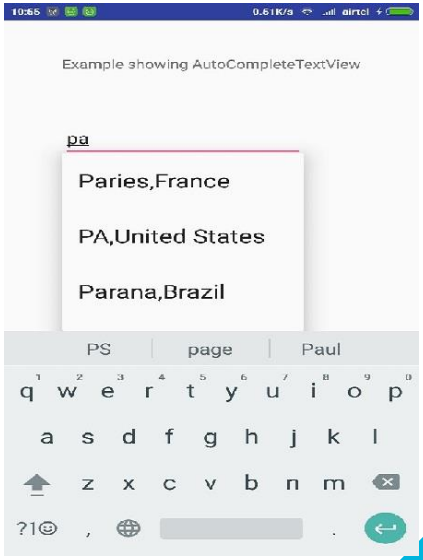
In android, TextView is a user interface control that is used to display the text to the user.

Android EditText

In android, EditText is a user interface control which is used to allow the user to enter or modify the text.

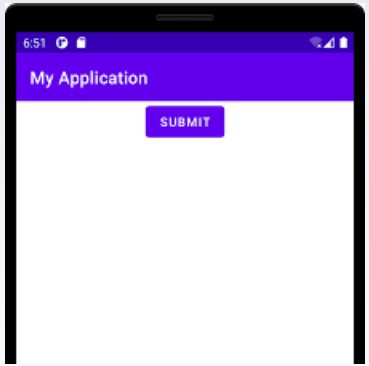
Android AutoCompleteTextView

In android, AutoCompleteTextView is an editable text view which is used to show the list of suggestions based on the user typing text. The list of suggestions will be shown as a dropdown menu from which the user can choose an item to replace the content of the textbox.



Android Button

In android, Button is a user interface control that is used to perform an action when the user clicks or tap on it.

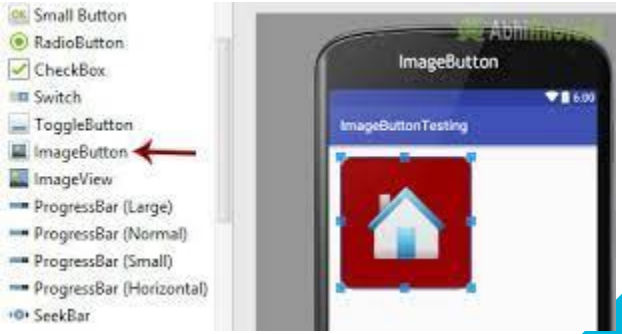


Android Toggle Button

In android, **Toggle Button** is a user interface control that is used to display ON (Checked) or OFF (Unchecked) states as a button with a light indicator.

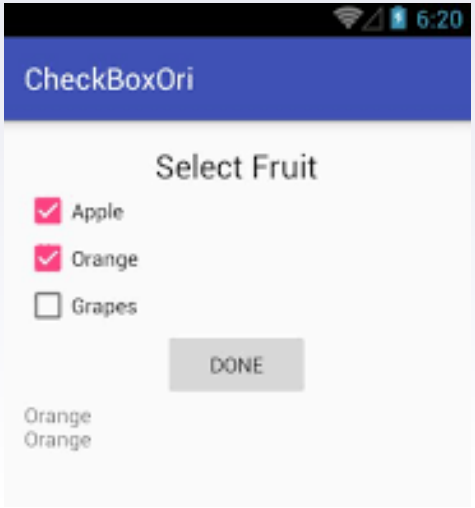
Android Image Button

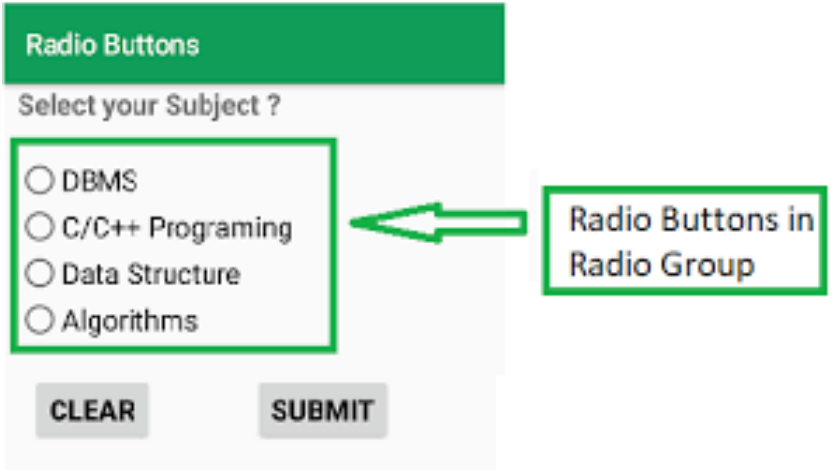
In android, Image Button is a user interface control that is used to display a button with an image to perform an action when the user clicks or tap on it.
Generally, the Image button in android looks similar as regular Button and perform the actions same as regular button but only difference is for image button we will add an image instead of text.



Android CheckBox

In android, Checkbox is a two-states button that can be either checked or unchecked.





Android Radio Group

In android, Radio Group is used to group one or more radio buttons into separate groups based on our requirements.

In case if we group radio buttons using the radio group, at a time only one item can be selected from the group of radio buttons.

Android Radio Button

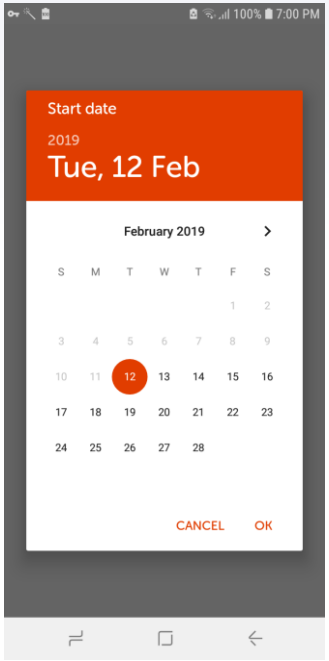
In android, Radio Button is a two-states button that can be either checked or unchecked and it cannot be unchecked once it is checked.

Android ProgressBar

In android, ProgressBar is a user interface control which is used to indicate the progress of an operation.

Android DatePicker

In android, DatePicker is a widget for selecting a date.

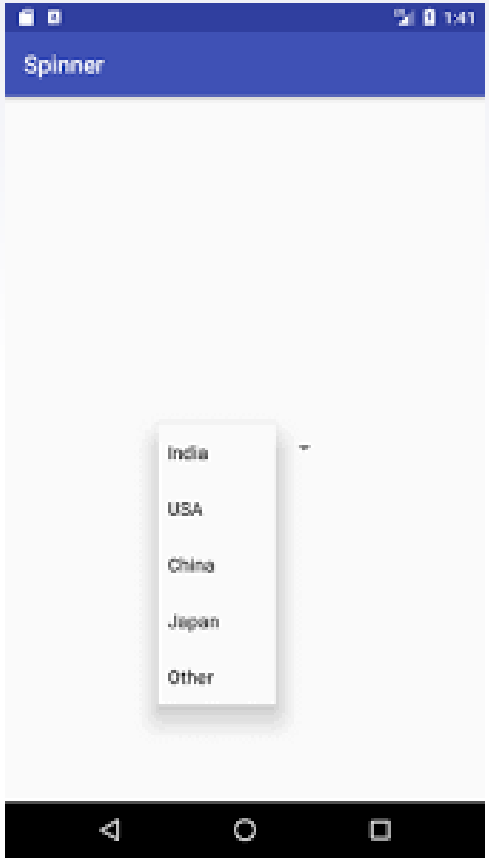


Android TimePicker

In android, TimePicker is a widget for selecting the time of day, either in 24-hour or AM/PM mode.

Android Spinner

In android, Spinner is a drop-down list which allows a user to select one value from the list.

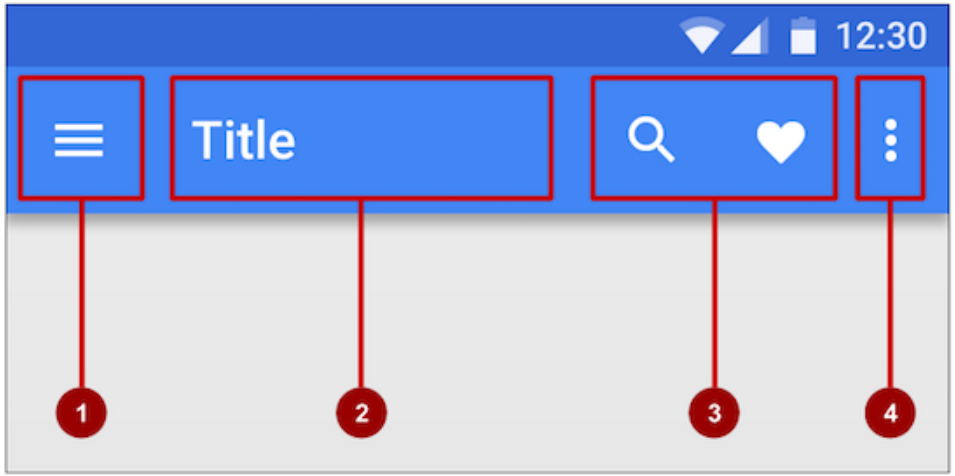


User Interface(UI) Design In Android

The typical UI of any Android application consists of these components:

- Main Action Bar (MAR)
- Split Action Bar (SAB)
- Content Area

These play a major role while you are developing a complex application.



1. *Navigation button or Up button:* Use a navigation button in this space to open a navigation drawer, or use an Up button for navigating up through your app's screen hierarchy to the parent activity. Both are described in the next chapter.
2. *Title:* The title in the app bar is the app title, or the name defined in AndroidManifest.xml by the `android:label` attribute for the activity.
3. *Action icons for the options menu:* Each action icon appears in the app bar and represents one of the options menu's most frequently used items. Less frequently used options menu items appear in the overflow options menu.
4. *Overflow options menu:* The overflow icon opens a popup with option menu items that are not shown as icons in the app bar.

Android widget

- Widgets display glanceable information of an app's most important data and functionality.

Types

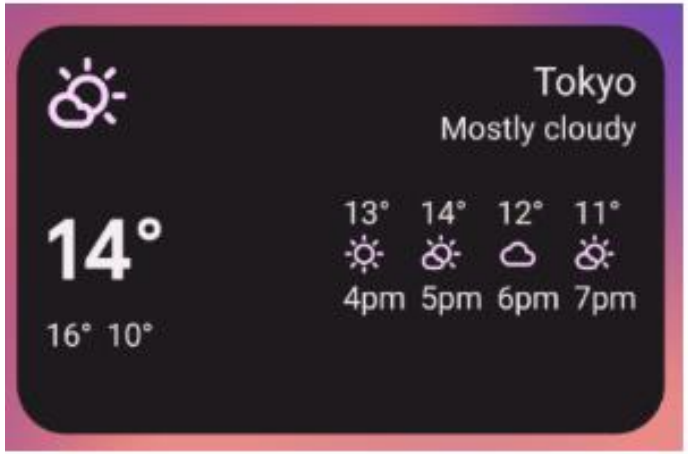
- Widgets display your app's new and interesting content in a consolidated form on a mobile home screen. They link to richer detail within the app.

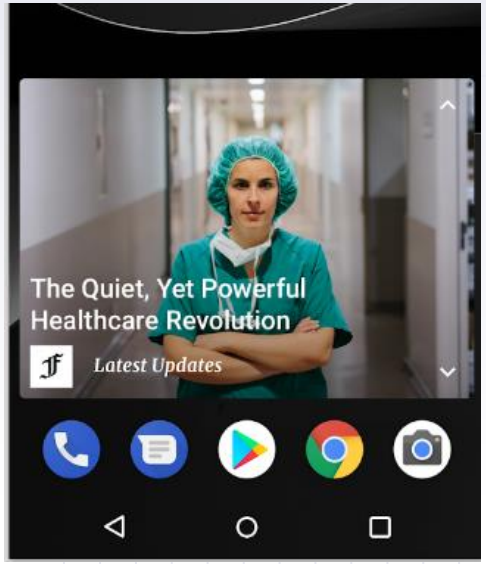
- Users can move and, if supported, resize widgets across their home screen panels.

- Information widgets
- Collection widgets
- Control widgets
- Hybrid widgets

Information widgets

Information widgets display a few elements of importance to a user and track how that information changes over time, such as weather or sports scores. Tapping the widget launches the associated app into a detail screen.



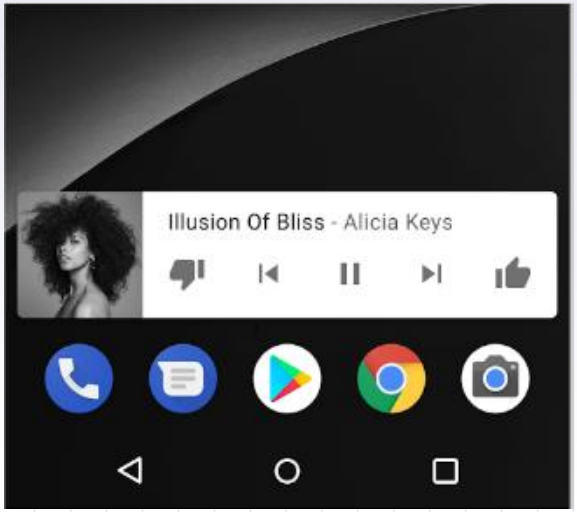


Collection widgets

Collection widgets display multiple elements of the same type, such as a collection of articles from a news app. They focus on two interactions:

- Browsing a collection
- Opening an element's detail screen

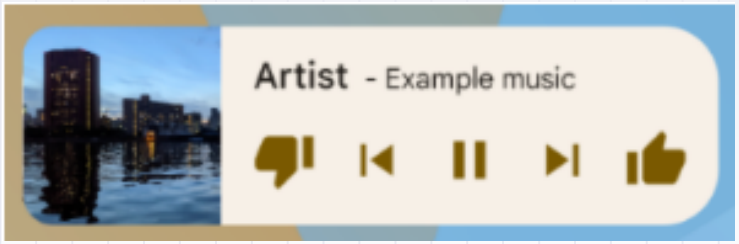
Collection widgets can scroll vertically.



Control widgets

Control widgets display frequently used functions. These functions may be triggered from the home screen without opening the app. For example, music app widgets allow the user to play, pause, or skip music tracks from outside the music app.

Control widgets may or may not progress to a detail screen.



Hybrid widgets

Many widgets are hybrids that combine elements of the different types above. Center your widget around one of these types and add elements of others as needed.

Behavior

Scrolling

Scrollable widgets

List or grid-based collection widgets usually expand or contract the vertical scrolling area. Regardless of the widget's size, the user can still scroll all elements into view.

Determine how much of your app's information should surface. For smaller widget sizes, concentrate on the essential and then add more contextual information as the widget grows.

Non-scrollable widgets

Information widgets are not scrollable. All content and layout must dynamically fit into the size selected by the user.

Responsive resizing

Widgets should accommodate different spacing requirements across devices, including cell number, size, and spacing variations.

Resizing

Resizing allows users to adjust the height or width of a widget. This allows users to influence the layout of widgets on home panels.

Your app may be completely resizable or constrained to horizontal or vertical size changes.

A long press and subsequent release sets resizable widgets into resize mode. Users can use the drag handles or the widget corners to set the desired size.

Navigation

Your widgets should provide navigation links to frequently used areas of your app, including:

- Functions that allow the user to create new content, such as a new document or message
- Access to the top level of your app

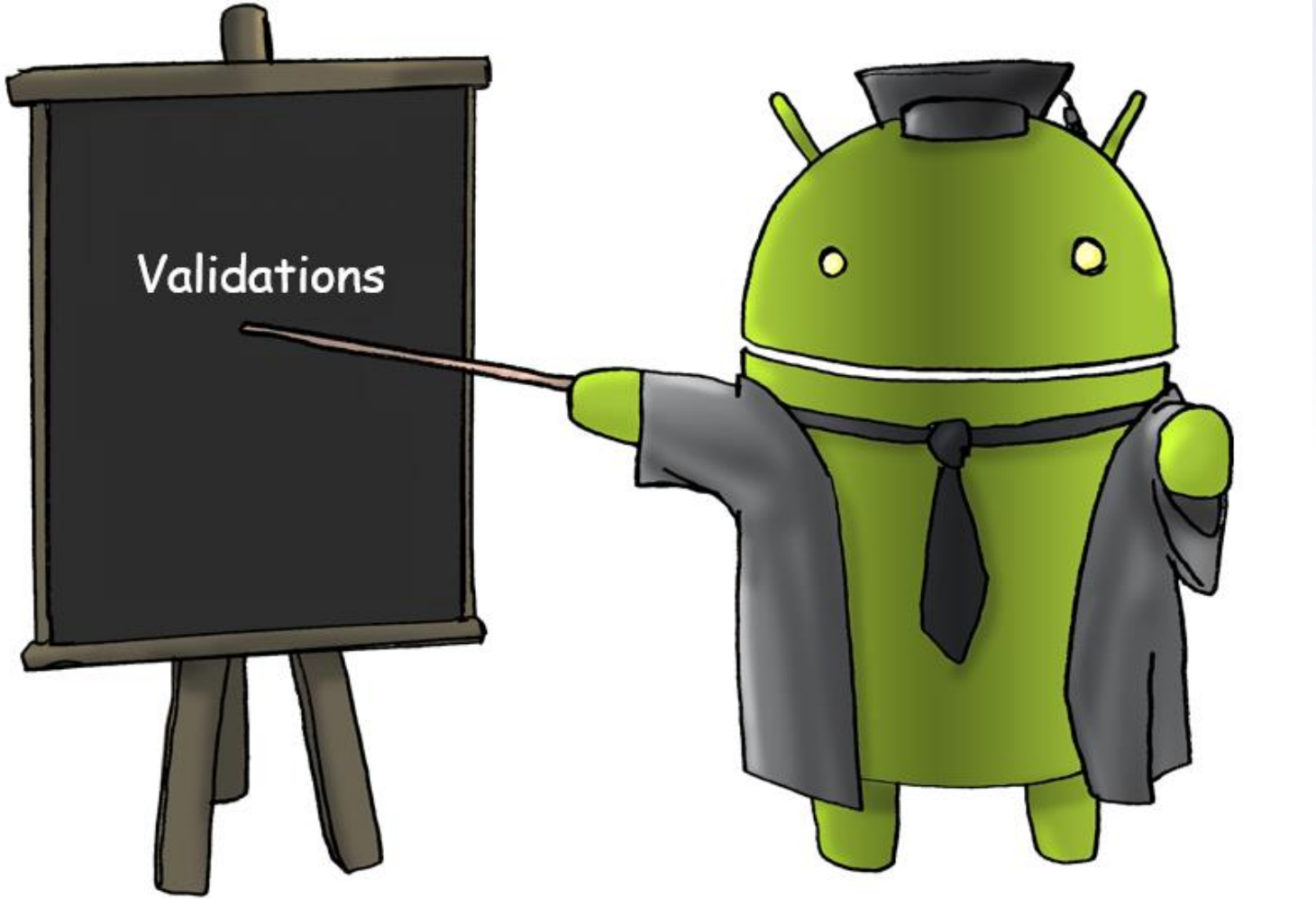
Configuring

Once placed on a home screen panel, Android widgets display their configuration choices.

Configuration should:

- Present no more than 2-3 configuration elements
- Present choices using dialogs, rather than full-screen, to retain the user's context

Once set up, widgets do not typically show a "Setup" or "Configuration" button.


Validations

FORM VALIDATION IN ANDROID STUDIO

- Validation need to ne done individually for each field and create separate function to perform validation.
- We will call all the functions from the main function or the function which only executes when user presses the button.
- So we will use the code below to call each function with single to make sure that each function should be executed inside if() statement.

INSIDE ONCLICK FUNCTION

```
if (!validateFullName() | !validateUsername() | !validateEmail() | !validatePassword()) {
    return;
}
```

VALIDATE FULLNAME

```
private boolean validateFullName() {
  String val = fullName.getEditText().getText().toString().trim();
  if (val.isEmpty()) {
      fullName.setError("Field can not be empty");
      return false;
  } else {
      fullName.setError(null);
      fullName.setErrorEnabled(false);
      return true;
  }
```

VALIDATE USERNAME

```
private boolean validateUsername() {
  String val = username.getEditText().getText().toString().trim();
  String checkspaces = "Aw{1,20}z";

  if (val.isEmpty()) {
      username.setError("Field can not be empty");
      return false;
  } else if (val.length() > 20) {
      username.setError("Username is too large!");
      return false;
  } else if (!val.matches(checkspaces)) {
      sername.setError("No White spaces are allowed!");
      return false;
  } else {
      username.setError(null);
      username.setErrorEnabled(false);
      return true;
  }
}
```

VALIDATE EMAIL

```
private boolean validateEmail() {
    String val = email.getEditText().getText().toString().trim();
    String checkEmail = "[a-zA-Z0-9._-]+@[a-z]+.+[a-z]+";

    if (val.isEmpty()) {
        email.setError("Field can not be empty");
        return false;
    } else if (!val.matches(checkEmail)) {
        email.setError("Invalid Email!");
        return false;
    } else {
        email.setError(null);
        email.setErrorEnabled(false);
        return true;
    }
}
```

VALIDATE PASSWORD

```
private boolean validatePassword() {
    String val = password.getEditText().getText().toString().trim();
    String checkPassword = "^" +
            //"(?=.*[0-9])" +         //at least 1 digit
            //"(?=.*[a-z])" +         //at least 1 lower case letter
            //"(?=.*[A-Z])" +         //at least 1 upper case letter
            "(?=.*[a-zA-Z])" +      //any letter
            //"(?=.*[@#$%^&+=])" +    //at least 1 special character
            "(?=S+$)" +           //no white spaces
            ".{4,}" +               //at least 4 characters
            "$";

    if (val.isEmpty()) {
        password.setError("Field can not be empty");
        return false;
    } else if (!val.matches(checkPassword)) {
        password.setError("Password should contain 4 characters!");
        return false;
    } else {
        password.setError(null);
        password.setErrorEnabled(false);
        return true;
    }
}
```

VALIDATE GENDER

```
private boolean validateGender() {
    if (radioGroup.getCheckedRadioButtonId() == -1) {
        Toast.makeText(this, "Please Select Gender", Toast.LENGTH_SHORT).show();
        return false;
    } else {
        return true;
    }
}
```

VALIDATE AGE

```
private boolean validateAge() {
    int currentYear = Calendar.getInstance().get(Calendar.YEAR);
    int userAge = datePicker.getYear();
    int isAgeValid = currentYear - userAge;

    if (isAgeValid < 14) {
        Toast.makeText(this, "You are not eligible to apply", Toast.LENGTH_SHORT).show();
        return false;
    } else
        return true;
}
```

VALIDATE PHONE NUMBER

```
private boolean validatePhoneNumber() {
    String val = phoneNumber.getEditText().getText().toString().trim();
    String checkspaces = "Aw{1,20}z";
    if (val.isEmpty()) {
        phoneNumber.setError("Enter valid phone number");
        return false;
    } else if (!val.matches(checkspaces)) {
        phoneNumber.setError("No White spaces are allowed!");
        return false;
    } else {
        phoneNumber.setError(null);
        phoneNumber.setErrorEnabled(false);
        return true;
    }
}
```

What Is Preference File?

- A preference file is actually a xml file saved in internal memory of device.
- Preferences in Android are key-value pairs used to store user's data with the flow of the application. The data exists even after closing the application. However, the data gets removed when the user uninstalls or clears app data. Preferences are usually used to store data of small size.
- Preferences are pretty helpful while storing user details on the go.
- Preferences are basically used to keep track of the application along with the user preferences.
- A preference is a simple key-value pair with a specific data type for the value. The task of preference key is to uniquely identify a preference and on the other hand values store the preference values.

Usually, preferences support only primitive data types like as follows:

- Boolean Types
- Integer Types
- Float Types
- String Types
- Long Types

Note: While storing data in preferences, you should ensure a unique key is assigned for each value. This key is later on used to fetch the data from the preferences.

Suppose someone installed your application, and you want them to fill in their details before using the application.

Now, these details like name, mobile number, address can be stored in the preferences and shown to the user whenever required.

APIs involved in accessing Preferences

Currently, we have three kinds of APIs that can be used to access preferences in any android application.

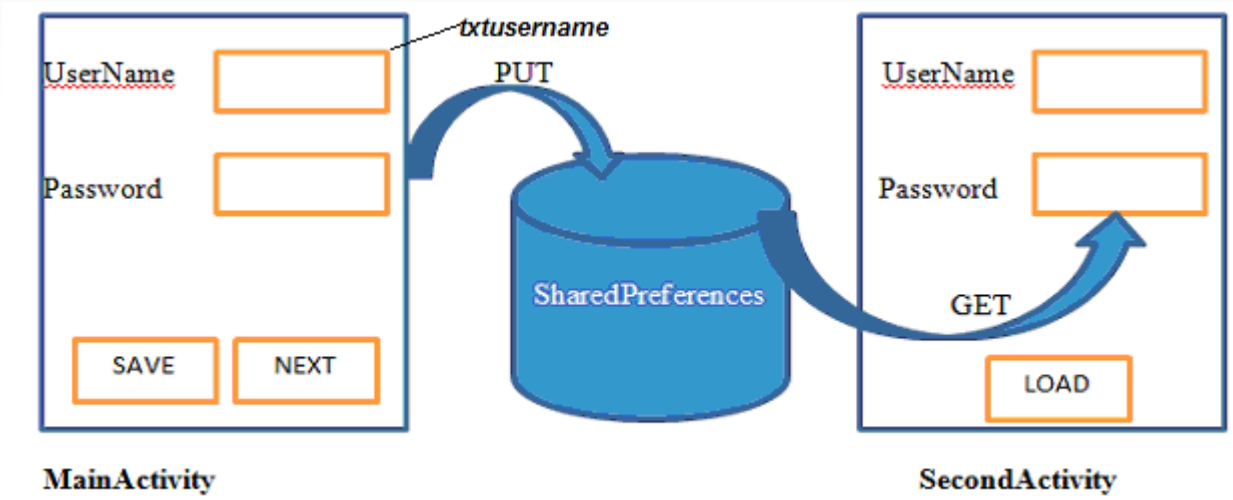
These are:

1. getPreference() – The getPreference() is used to access activity-specific preferences.

2. getSharedPreference() – The getSharedPreference() is used to access application-level preferences. Using getSharedPreference() you can access data from other activities as well as other application contexts.

3. getDefaultSharedPreference() – The getDefaultSharedPreference() function returns the Android preference framework's default SharedPreference. We can also use this to access SharedPreference in Fragments.

Shared Preferences



What are SharedPreferences?

- SharedPreferences is used by apps to save data in name-values pairs, like a Bundle
- Data is stored in XML file in the directory data/data/<package-name>/shared-prefs folder
- Store data such as username, password, theme settings, other application settings
- SharedPreferences only allows you to save primitive data types (that is, booleans, floats, longs, ints, and strings)

key	value
firstName	Bugs
lastName	Bunny
location	Earth

- The DATA folder can be obtained by calling Environment.getDataDirectory(). SharedPreferences is application specific, i.e. the data is lost on performing one of the following options:

  on uninstalling the application
  on clearing the application data (through Settings)

- Shared Preferences, as the name suggests, is a storage that is shared throughout your application.
- It implies that the key-value pairs stored as preferences are available to all the activities, fragments, and other application contexts present within the app.
- To use Shared Preferences in a particular activity, we first need to get the instance of it.
- Shared Preference comes with two words, Shared and Preferences. Here, Share means distributing the data and Preferences here means important or something that is preferable.

How to access them?

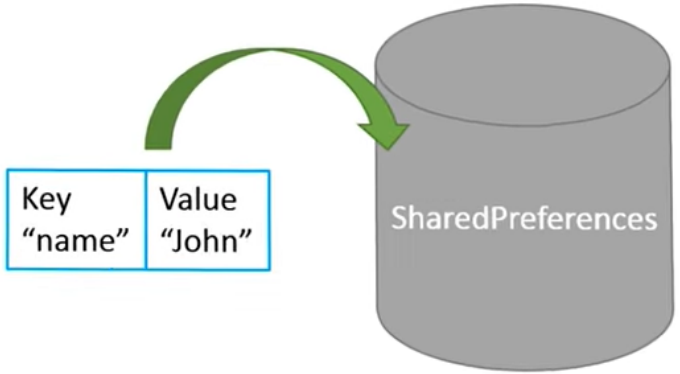
```
SharedPreferences sharedpreferences = getSharedPreferences( MyPREFERENCES, Context. MODE_PRIVATE);
```

- If you have only 1 preference file, call **getPreferences(int mode)**
- If you have several files call **getSharedPreferences(String name, int mode)**
- MODE_PRIVATE: Only your app can access the file
- MODE_WORLD_READABLE: All apps can read the file ✖
- MODE_WORLD_WRITEABLE: All apps can write to the file ✖
- MODE_MULTI_PROCESS: Multiple processes can modify the same shared preference file

How to use SharedPreferences?

To Store data

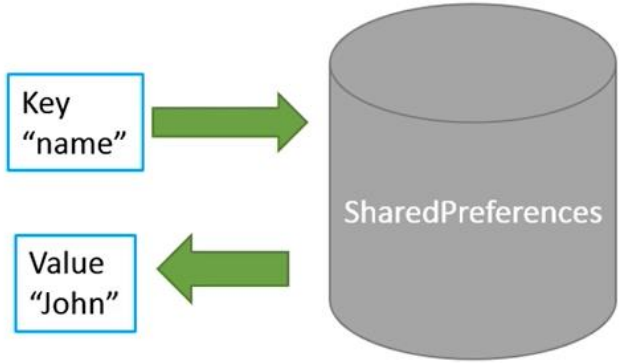
1. Get a reference to the SharedPreferences object
    1. For a single file, call **getPreferences(int mode)**
    2. For several files, call **getSharedPreferences(String name, int mode)**
2. Call the editor
3. Use the editor to add the data with a key
4. **Commit editor changes**



```
// Storing data into SharedPreferences (Initialization)
SharedPreferences sharedPreferences = getSharedPreferences("MySharedPref",MODE_PRIVATE);

// Creating an Editor object to edit(write to the file)
SharedPreferences.Editor myEdit = sharedPreferences.edit();

// Storing the key and its value as the data fetched from edittext
myEdit.putString("name", name.getText().toString());
myEdit.putInt("age", Integer.parseInt(age.getText().toString()));

// Once the changes have been made,we need to commit to apply those changes made, otherwise, it will throw an error
myEdit.commit();
```

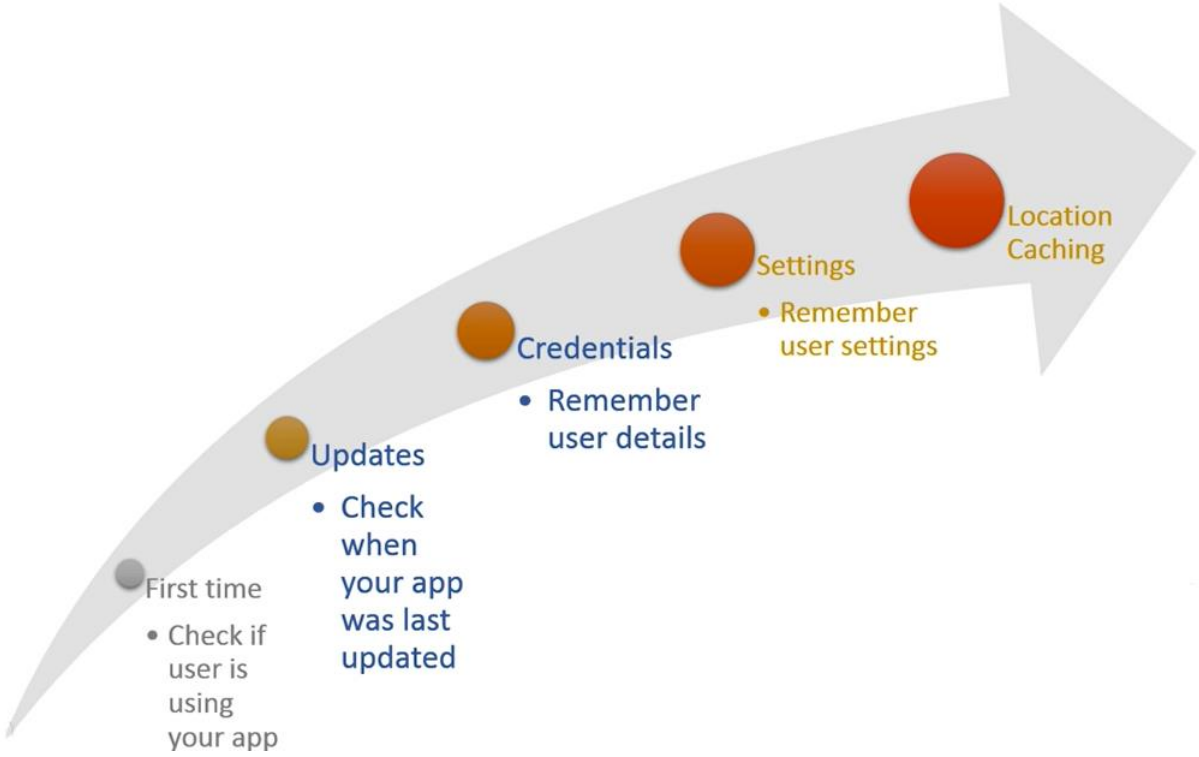
To retrieve data

1. Get a reference to the SharedPreferences object
    1. For a single file, call **getPreferences(int mode)**
    2. For several files, call **getSharedPreferences(String name, int mode)**
- Use the key provided earlier to get data
- **Supply default values if the data is not found**

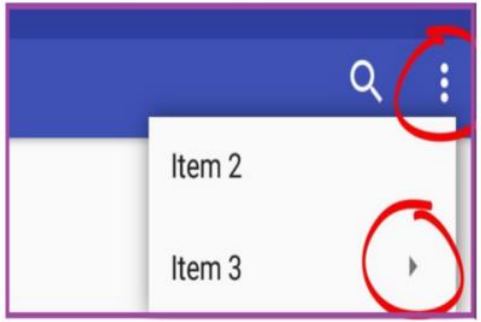


```
SharedPreferences sharedpreferences = getSharedPreferences(String PREFS_NAME, Context.MODE_PRIVATE);

Editor editor = sharedpreferences.edit();

 sharedpreferences.getString("key_name", null);      // getting String
 sharedpreferences.getInt("key_name", -1);           // getting Integer
 sharedpreferences.getFloat("key_name", null);       // getting Float
 sharedpreferences.getLong("key_name", null);        // getting Long
 sharedpreferences.getBoolean("key_name", null);   // getting Boolean

editor.remove("name");                               // will delete key name
editor.remove("email");                              // will delete key email

editor.commit();                                     // commit changes

editor.clear();                                      // REMOVE ALL DATA!

editor.commit();                                     // commit changes
```

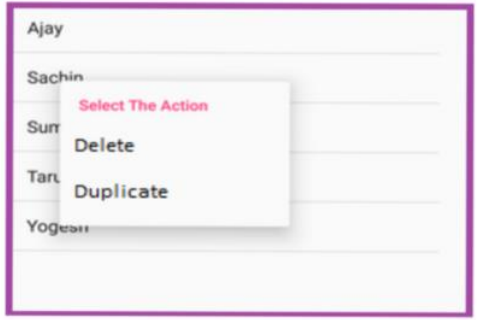
Some uses of SharedPreferences



Options Menu



Contextual Menu



Popup Menu

Menus In Android

- ✓ In android, Menu is an important part of the UI component which is used to provide some common functionality around the application.
- ✓ With the help of menu, users can experience a smooth and consistent experience throughout the application.
- ✓ In order to use menu, we should define it in a separate XML file and use that file in our application based on our requirements.
- ✓ Also, we can use menu APIs to represent user actions and other options in our android application activities.

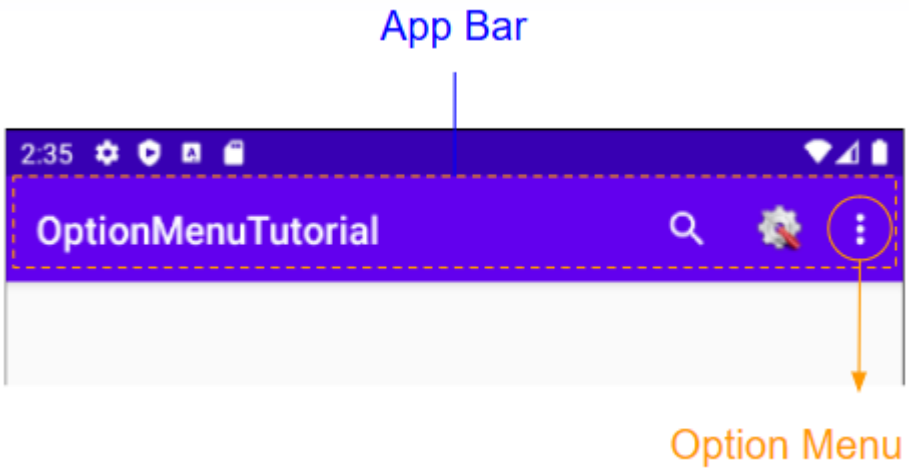
- ✓ There are 3 types of menus in Android:

- Option Menu
- Context Menu
- Pop-up Menu

1- Android Option Menu

In Android, an Option Menu is a set of primary options of an application which users can select one of the options to perform an action. The Option Menu appears on the right side of the App Bar.

The options menu is the primary collection of menu items for an activity. It's where you should place actions that have a overall impact on the app, such as Search, Compose Email and Settings.



2 - Android Context Menu

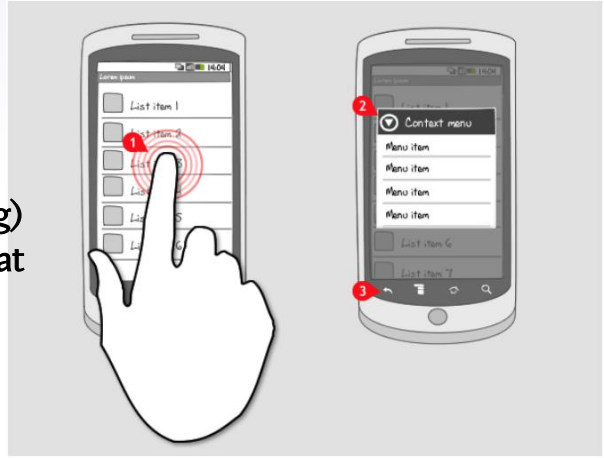
A context menu is a floating menu that appears when the user performs a long-click on an element.

A contextual menu offers actions that affect a specific item or context frame in the UI.

There are two ways to provide contextual actions:

In a floating context menu.

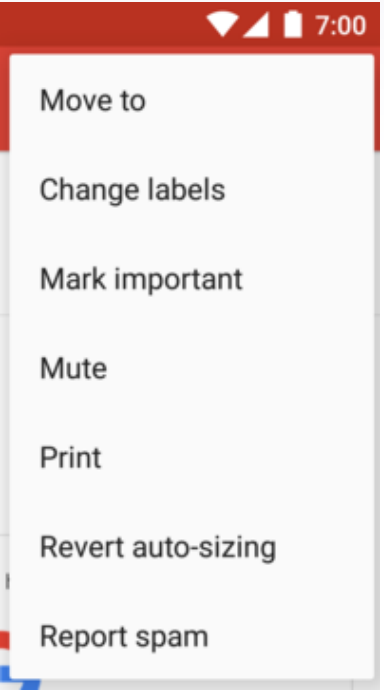
A menu appears as a floating list of menu items (similar to a dialog) when the user performs a long-click (press and hold) on a view that declares support for a context menu. Users can perform a contextual action on one item at a time.



In the contextual action mode

This mode is a system implementation of ActionMode that displays a contextual action bar at the top of the screen with action items that affect the selected item(s). When this mode is active, users can perform an action on multiple items at once (if your app allows it).

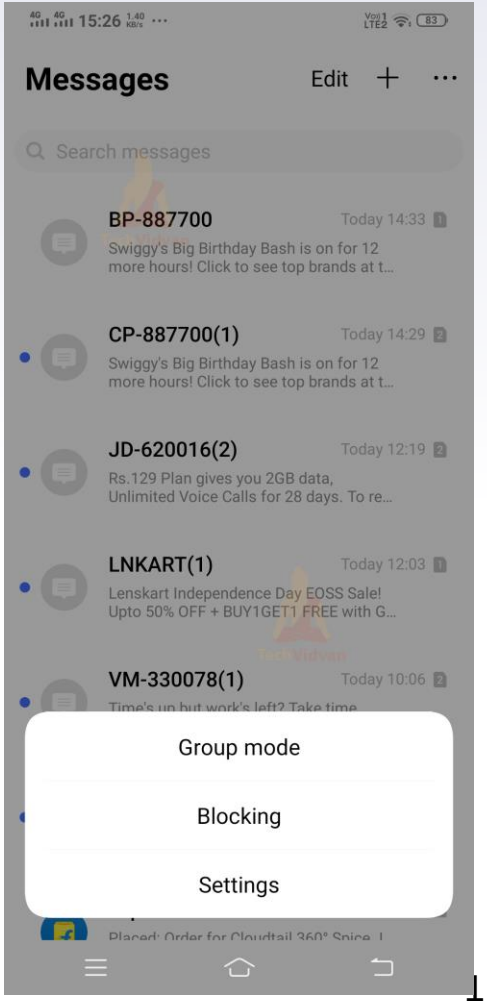
3 - Android PopUp Menu



A popup menu displays a list of items in a vertical list that is anchored(sticked) to the view that invoked the menu.

It's good for providing an overflow of actions that relate to specific content or to provide options for a second part of a command.

Note: Actions in a popup menu should not directly affect the corresponding content—that's what contextual actions are for. Rather, the popup menu is for extended actions that relate to regions of content in your activity.



Following are important elements of a menu:

<element>	Description
<menu>	1. It defines a Menu, that contains the options 2. Holds one or more elements 3. It must be the root of a file.
<items>	1. Items are the options that are present in the Menu 2. It must be in the menu element 3. It can contain a nested <menu> element to create a submenu.
<group>	1. Categorizes the menu items and contains those with common properties. 2. It is optional and invisible container for elements.

Like any other UI component, even Android menus can be customized with the help of attributes like-

1. android: id - It uniquely identifies the item of the menu.

2. android: icon - It sets an icon to represent the item.

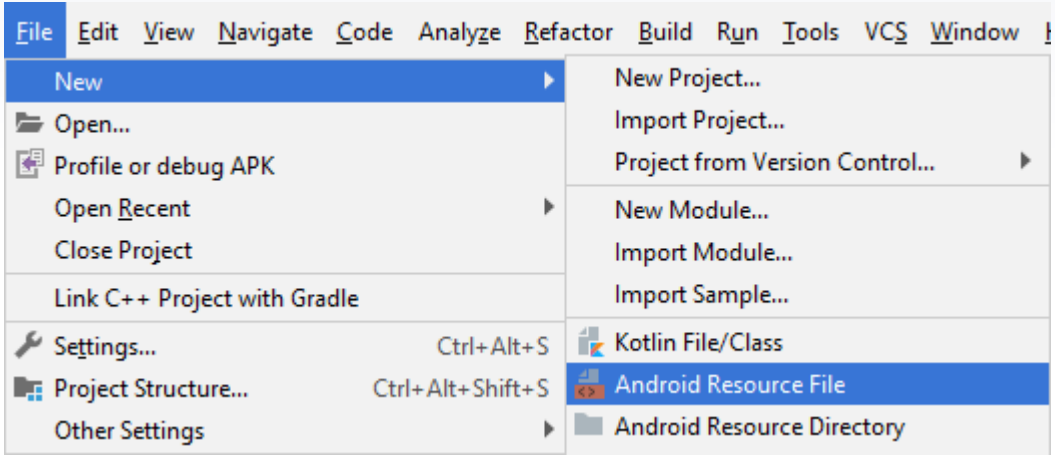
3. android: title - It sets the title of the item.

4. android:showAsAction - It specifies when and how this item should appear as an action item in the app bar. The value can be *ifRoom, never, withText, always, collapseActionView*.

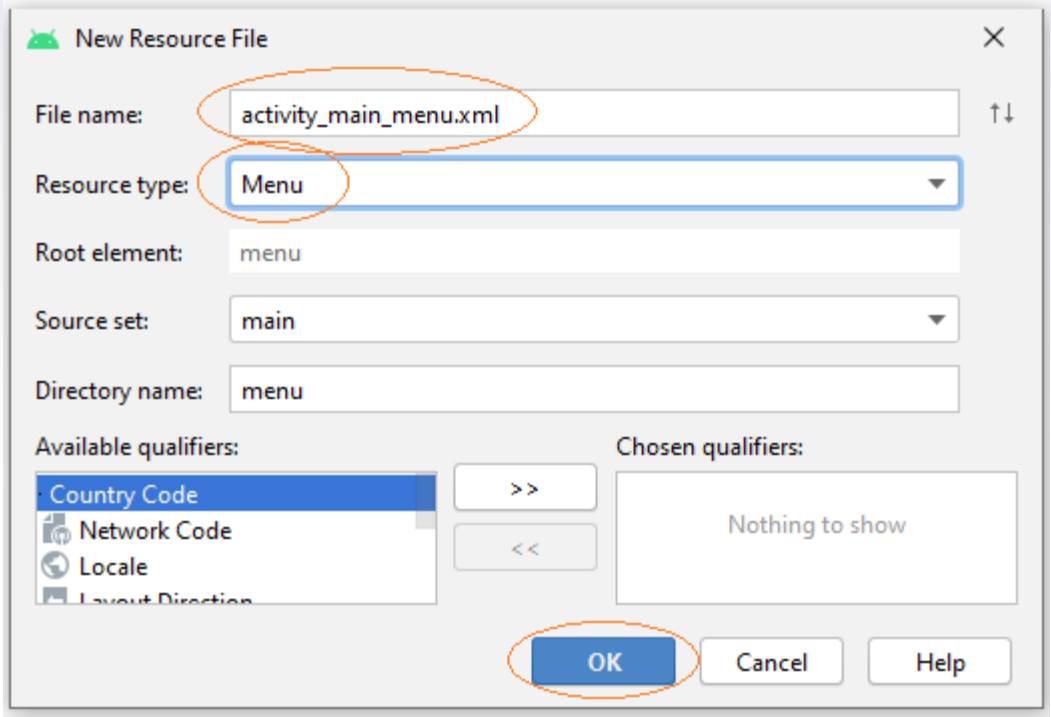
Implementation of Android Option Menu

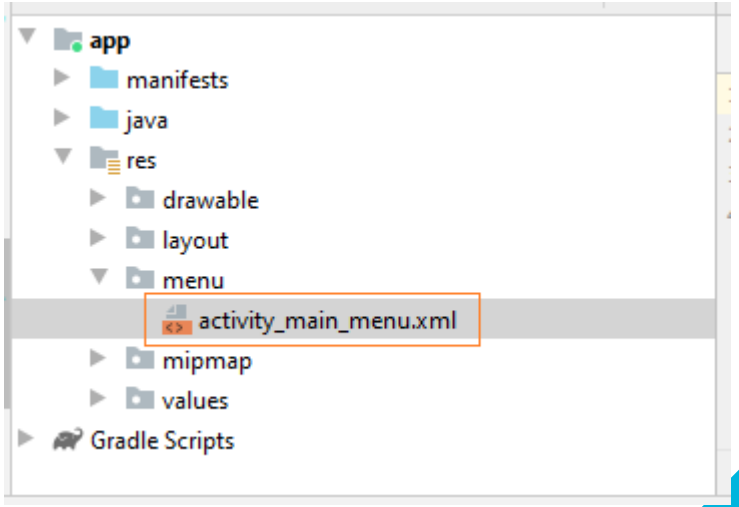
1. First, create a new project and name it.

2. On Android Studio, select: File > New > Android Resource File



- File name: activity_main_menu.xml
- Resource Type: Menu


New Resource File
File name:
activity_main_menu.xml
Resource type:
Menu
Root element:
menu
Source set:
main
Directory name:
menu
Available qualifiers:
Country Code
Network Code
Locale
>>
<<
Chosen qualifiers:
Nothing to show
OK
Cancel
Help


app
manifests
java
res
drawable
layout
menu
activity_main_menu.xml
mipmap
values
Gradle Scripts

res/values/strings.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Concrete Page</string>
    <string name="message">Click on menu for Menu Demo</string>
    <string name="bookmark">Bookmark</string>
    <string name="save">Save</string>
    <string name="search">Search</string>
    <string name="share">Share</string>
    <string name="delete">Delete</string>
    <string name="print">Print</string>
</resources>
```

res/layout/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"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#C98C00"
    tools:context=".MainActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/message"
        android:textSize="25sp"/>
</LinearLayout>
```

res/menu/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/bookmark_menu"
          android:title="@string/bookmark"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/save_menu"
          android:title="@string/save"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/search_menu"
          android:title="@string/search"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/share_menu"
          android:title="@string/share"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/delete_menu"
          android:title="@string/delete"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/print_menu"
          android:title="@string/print"
          android:showAsAction="ifRoom"/>
</menu>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.concretepage"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="16"/>
    <application
        android:allowBackup ="false"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <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>
```

To create menu we have to override onCreateOptionsMenu, in which we use getMenuInflater().inflate that inflates a menu hierarchy from XML resource. To handle click event, override onOptionsItemSelected in Activity class.

Create Menu Item in XML

Create menu Item in XML file within *res/menu*.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/bookmark_menu"
          android:title="@string/bookmark"
          android:showAsAction="ifRoom"/>
</menu>
```

<menu>: Root node that contains one or more item.

<item>: It represents menu items.

android:id: Unique ID for the item.

android:title : Title for the item.

android:showAsAction : It specifies how an item to be shown as an action item. The value can be *ifRoom, never, withText, always, collapseActionView*.

Override onCreateOptionsMenu and use getMenuInflater().inflate

Find the `onCreateOptionsMenu(Menu menu)` method which needs to override in `Activity` class. This creates menu and returns Boolean value. `inflate` inflates a menu hierarchy from XML resource.

```
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
```

Override onOptionsItemSelected(MenuItem item) to Handle Click Event

To handle click event on menu item, we have to implement `onOptionsItemSelected(MenuItem item)` in `Activity` class and return Boolean value. For the demo we are using `Toast` to display click event.

```
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
            case R.id.bookmark_menu:
                Toast.makeText(this, "You have selected Bookmark Menu", Toast.LENGTH_SHORT).show();
                return true;
    }
}
```

After that write the following code in the MainActivity.java file:

```
package com.example.optionmenuapplication;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.options_menu, menu);
    return super.onCreateOptionsMenu(menu);
  }

  @Override
  public boolean onOptionsItemSelected(@NonNull MenuItem item) {
    Toast.makeText(this, "Selected Item: " + item.getTitle(),
Toast.LENGTH_SHORT).show();
    switch (item.getItemId()) {
      case R.id.search_item:
        // do your code
        return true;
      case R.id.upload_item:
        // do your code
        return true;
      case R.id.copy_item:
        // do your code
        return true;
      case R.id.print_item:
        // do your code
        return true;
      case R.id.share_item:
        // do your code
        return true;
      case R.id.bookmark_item:
        // do your code
        return true;
      default:
        return super.onOptionsItemSelected(item);
    }
  }
}
```

Thank You!