**What is android?**

android is a software package and linux based operating system used for mobile devices such as tablets , smartphones and computers.

Android was developed by the Open Handset Alliance, led by Google, and other companies java language is mainly used to write android code.

The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.

The goal of android project is to create a successfull real world project that improves the mobile experinces for end users.

**Why Android ?**



## Features of Android :

1. UI Design
2. Connectivity
3. Media Support
4. Storage
5. Messaging
6. WebBrowser
7. Multitouch
8. Multilanguage etc.

**Hierarchies of Android :**

|  |  |  |
| --- | --- | --- |
| **Version** | **Code name** | **API Level** |
| 1.5 | Cupcake | 3 |
| 1.6 | Donut | 4 |
| 2.1 | Eclair | 7 |
| 2.2 | Froyo | 8 |
| 2.3 | Gingerbread | 9 and 10 |
| 3.1 and 3.3 | Honeycomb | 12 and 13 |
| 4.0 | Ice Cream Sandwitch | 15 |
| 4.1, 4.2 and 4.3 | Jelly Bean | 16, 17 and 18 |
| 4.4 | KitKat | 19 |
| 5.0 | Lollipop | 21 |

**Android Setup Environment prerequests :**

1. Android studio 1.5.
2. jdk 1.8

**Android Project Structure Details:**

1. Android Folders and Files.
2. Android Application Module.

**Android Folders and Files :**

1. . idea
2. . app
3. . build
4. . Gradle
5. . git ignore
6. build . gradle
7. gradle . properties
8. gradlew
9. gradlew.bat
10. local properties
11. . iml
12. settings . gradle
13. . idea : it is provided by the android studio intelli-J IDEA.

In the .idea we add plugins and configuration with respect to files .

The .idea is autogenerated.

1. app : It is an application that contains modules and files

ex: name of the project is application and contents inside it are modules and files.

1. . build : It is a directory store and build output for the project modules.
2. gradle : It is a build system which is used for building of app , gradle contains gradled

wrapper files.

1. . gitignore : It is related to github i.e. push and pull the code in repository , specified the untracked files that git should ignore.
   * 1. pull / checkout : It is used to get the file from repository to local machine for editing and uploading it with same url.
     2. push / checkin : It is used to store files from local machine to repository.
2. build . gradle : It contains customizable properties for the build system.
3. gradle . properties : It contains project wise gradle settings ex : while using irctc app we need to use banking app for transaction so both the application properties are needed.
4. gradlew : It is used to run the project in linux , it is a gradle startup script for linux.
5. gradlew . bat : gradle startup script for windows.
6. local properties : It contains SDK installation details.
7. . iml : It contains module file created by intelli-J IDE to store module information.
8. settings . gradle : It specifies the sub project to build.

**Android Application Module :**

1. build /
2. libs /
3. src /
4. androidtest /
5. main / gen
6. main / res
7. anim /
8. color /
9. drawable /
10. layout /
11. minmap /
12. menu /
13. values /
14. xml /
15. AndroidManifest.xml /
16. build / : It contains folder for the specified build variants.
17. libs / : It contains private libraries stored in the main application module.
18. src / : It contains stubbed activity files (. java classes) ex: src/main/java/androiddemo.java
19. AndroidTest / : It contains instrumentation tests.
20. main / gen : It contain java files generated by Android studio such as your R.java file.
21. main / res : It contains applications Resources such as drawable file , layout files, string files etc.
22. anim / : It contains all Animation objects.
23. color / : It is an xml file that describe colors.
24. drawable / : It contains bitmap file(png,jpeg, mpeg,gif).
25. layout / : it is an xml file that are compiled into screen layout and it is part of screen.
26. minmap / :It contains launcher icons.
27. menu / : It is an xml file that defines application menus.
28. values / : xml file that define resources by xml element types.
29. xml / : miscelleneous xml file that configure application components.
30. AndroidManifest.xml / : The control file that describe the nature of the application and each of its components such as activities , services , intents Receivers and content providers.

* what permissions are requested.
* what external libraries are needed.
* what device features are required.
* what API level are supported or required.

**Building an Android Application :**

compilation & Packing

Android Project

Device

or

Emulator

**hfjgjfg**

.dex files

Resource.arsc

uncomplied resources

Android Manifest.xml

Android build process provides project and modules into build settings so that your android modules are compiled and packged into .apk files the container for your application binary is based on your build settings.

**The** apk file for each app contains all of the information necessary to run your application on device or emulator such as compile.dex (.class files converted to Dalvik byte code ) binary version of AndroidManifest.xml compiled resources (resource.arsc uncompiled resource files for your application).

To run your application (android appliaction) on a emulator or device the application can be used either debug or release mode.

Debug mode : suppose you want to signin your application that time we use Debug mode when we develop and test your application because build system uses debug key.

**Building Blocks of Android (or) Android Components (or) Android Fundamentals :**

1. Activities
2. Services
3. Broadcast Service
4. Content Providers.

**Activities :**

* Activity represents a single screen with a userinterface.
* most Applications contains multiple activities.
* when a new activity starts it will be pushed onto back stack.
* user interface can built with xml or java.

**Services :**

* services perform Long running operation in the background.
* it does not contain user interface.
* usefull for things like network operations playing music etc.
* Runs independently of the component.
* Can be bound to by weather application components.

**Content Providers :**

* These are used to store and retreive data and make it accessible to all applications.
* It is the only way to share data across application .
* Exposes a public url that uniquely identifies its dataset.
* Data is exposed as a simple table on a database model.
* Android contains many providers like contacts , media etc.

**Broadcast Receivers :**

* A component that responds to system wide broadcast announcement ex:when the screen turnoff , battery is low.
* Application can also intiate their own broadcast .
* Broadcast Receiver contains no interfaces.

**Android Activity architecture :**

Here we are using project name as AndroidActivity and file name as mainactivity.

mainactivity : In mainactivity we Register button to perform action i.e..

1.we type cast button , use setOnClickListener method to handle the action , to invoke the functionality of button we invoke View class.

2.we use anonymous inner class i.e.. we start the activity by calling startActivity , we create a new intent object , then passing parameters as MainactivityActivity.this i.e..here this keyword refers to MainactivityActivity , demo.class contains demo.xml so first it will navigate from Mainactivity to Demo.class and hence it will display demo.xml contents.

mainactivity.xml :

It is the main xml file which will be created by default ,we declare Textview and create a button hence it is the first activity which appears on the screen

demo.xml :

It is a xml file in which we declare a text view i.e.. it becomes active when the button performs action.

* [onCreate(Bundle)](http://developer.android.com/reference/android/app/Activity.html#onCreate(android.os.Bundle)) is where you initialize your activity. Most importantly, here you will usually call [setContentView(int)](http://developer.android.com/reference/android/app/Activity.html#setContentView(int)) with a layout resource defining your UI, and using [findViewById(int)](http://developer.android.com/reference/android/app/Activity.html#findViewById(int)) to retrieve the widgets in that UI that you need to interact with programmatically.

**Notifications And Views :**

Notification is a message it can be used to display to the user outside of your application.

When you tell system to issue notification it first appears as an icon in the notification Area.

**Creating a Notification :**

NotificationCompact.Builder : It is an object extends class Object , It will create notification object and returns notification.

NotificationCompact.Builder.Build() : Build is a method of NotificationCompact.Builder.

To display notification we use method notify(). i.e...NotificationManager.notify().

**Notification Object Contain Following things :**

1. setsmallIcon() : display small icon.
2. setcontentTitle() : display title name.
3. setcontentTest() : display text.

Creating a Notification Object :

NotificationCompact.Builder notification = new NotificationCompact.Builder(this);

notification.setsmallIcon(R.layout.small\_icon);

notification.setcontentTitle("Display Notification");

notification.setcontentText("Welcome To NeoRays");

ex:

# small icon Title

Display Notifications

Welcome to I.T.Contractors

### Text

**notification** : Battery is low :- when it intimates the user may take decision instead the device shutdown directly.

**Removing The Notification :**

The user dismisses the notification individually or clear all.

The user clicks the notification and call setAutoCancel() method when you created the notification.

You can call Cancel() method for specific notification id this method also deletes ongoing notification.

You can call CancelAll() method which removes all the notifications.

**View :**

View is a part of notification.

notification will be visible (or) appear in two ways:

1. Normal view.
2. Big view.

The big view notification appears when the notification is expanded , this happens when the

notification is on the top of drawer (or) the user clicks the notification.

The ability to share (or) dismiss (delete) the notification.

**Create or Construct a View :**

Intent dismissIntent = new Intent (this , Pingservice.class);

dismissIntent.setAction(commonconstant.ACTION\_DISMISS);

PendingIntent penIntent = PendingIntent.getService(this , 0 , dismissIntent , 0);

**Layouts :**

A layout defines visual structure for a user Interface.

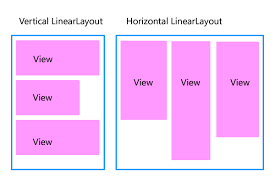
**Types of Layouts :**

1. Linear Layout.
2. Relative Layout.
3. Table Layout.
4. Frame Layout.
5. Absolute Layout.

**Linear Layout** :

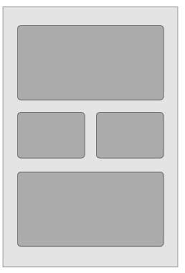
Linear Layout is a view group that aligns all the components in a single direction either vertical or

horizontal.



**Relative Layout** :

Relative Layout is a view group that displace child views in relative position.



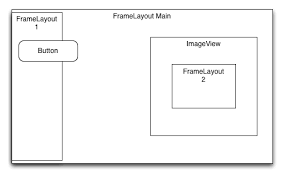
**Table Layout :**

Table Layout is a view that groups into rows and columns.



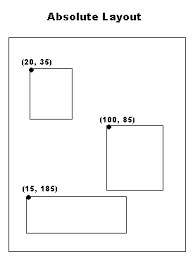
**Frame Layout** :

Frame Layout is placeholder onscreen that you can use to display single view.



**Absolute Layout :**

Absolute Layout enables view to specify the exact Location of its children.



**Activities and Activity Life Cycle :**

Activity is an application component that provides a screen with which user can interact in order to do something

such as take a photo (or) send an email (or) view a map.

Each activity have a window in which to draw its user interface.

**Activity life cycle :**

3 states in activity life cycle :

1. Resumed .
2. Paused.
3. Stop.

**Resumed** : In this state the activity is in foreground and the user can interact with it.

**Paused** : In this state the activity is in partially obscured by another activity , other activity that is in foreground is

semitransparent does not cover the entire screen.

**Stop** : In this state the activity is completely hidden or not visible to user , It is considered as background.

**Toast** :

* Toast is a solution for Android developer when required to notify user about an operation without expecting any user input.
* Toast is a small popup that displays for a small period and fades out automatically after timeout.
* Toast is a view containing a quick little message for the user.

**Creating a Toast** :

public class Toast extends Object

{

Toast.makeText(getApplicationContext," ",duration).show();

}

note : Toast class is present in android widget.package

**return type** **variable name** **description**

int Length\_LONG shows the view or notification for a long period of time.

int Length\_SHORT shows the view or notification for a short period of time.

**Parameters to passed inside toast** :

we can instantiate android.widget.Toast object using static method makeText() in this method pass 3 parameters:

1. applicationcontext.
2. Text Message.
3. Duration.

**Activity Life Cycle :**

onCreate()

onRestart()

onResume()

onStart()

onStop()

onPause()

onDestroy()

While Application moves through different states , the Android.app.Activity Life cycle method get called by system.

**onCreate() method** :

* It is called when the activity is created for first time we should intialize the data , create intial view.
* onCreate callback is always followed by onStart().

**onStart() method** :

* It is called when the activity is becoming visible.
* This is ideal place to write code that affects UI of application.
* This callback is normally followed by onResume() , but could be followed by onStop() ,if the Activity becomes hidden.

**onResume() method** :

* It is called when the Activity is running on the foreground and user can interact with it and is followed by onPause().

**onPause() method** :

* It is called when another Activity comes to the foreground.
* The implementation needs to be quick ,because the other Activity cannot run untill this method returns.
* The onPause() callback is followed by onResume().
* If the Activity returns to the foreground , or by onStop() if the activity becomes invisible.

**onStop() method** :

* It is called when the Activity is invisible to the user either a new Activity has started an exisiting Activity has resumed or this Activity getting destroyed.
* The onStop() callback is followed by onRestart() if the Activity returns to the foreground.

**onRestart() method** :

* It is called when the Activity is being restarted and when the Activity is returning to the foreground.
* It is always Followed by onStart().

**onDestroy() method** :

* It is called by the system before Activity is destroyed or either because the Activity is finishing or because the system is Reclaiming the memory the Activity is using.

**getcontext() method :**

* It returns the view is currently running in app usually the currently active Activity this method present in view object.

(or)

* Returns the context view only current running Activity.

**getApplicationContext() method :**

* This method returns the context for the entire application .

(or)

* Returns the context for all the Activities running in the application ,this method is present in Activity object.

**getBaseContext() method** :

* This method is present in context wrapper object.
* if you need access to a context from within another context (or) if you want to access context from another context within application you can use getBaseContext().

Note1 : context provides information about Activity or Application to newly created

components.

Note2 : context should be provided to newly created components (whether application context or

Activity context).

Note3 : Activity is a subclass of context.

**Android Messages : TO Send And Receive SMS**

Android offers full access to SMS functionality from our application with the SMS Manager.

SMS implements a mechanism that allows you to send text messages to other mobile for users irrespective of whether they have android phone.

SMS is supported by almost Everyone on earth so the latency is not a issue and updates are in frequent situation , SMS data messages are an excellent process.

**Sending SMS Messages :**

* Message services managing in Android is handled by the sms manager.
* you can get reference to the sms manager by using the static method i.e..SMSManager.getDefault().

step 1: Creating a Message

SMSManager smsManger=SMSManger.getDefault();

step 2 : In manifest.xml use following attribute to grant permission for sms i.e..

<uses-permission android:name="android.permission.SEND\_SMS"/>

Actually to send SMS Message our application requires the Send Permission .

Send Text Messages

our target to send info in a text message format use SendTextMessage() from the

SMSManager(class) passing in the phone number of receiver send the text message we want to

send.

Step 3 : String sendto = "9481663087"

String msg= "Hello"

# Text Fields :

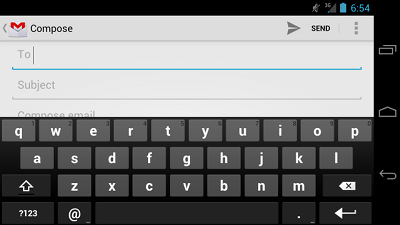
A text field allows the user to type text into your app. It can be either single line or multi-line. Touching a text field

places the cursor and automatically displays the keyboard. In addition to typing, text fields allow for a variety of

other activities, such as text selection (cut, copy, paste) and data look-up via auto-completion.

You can add a text field to you layout with the [EditText](http://developer.android.com/reference/android/widget/EditText.html) object. You should usually do so in your XML layout with

a <EditText> element.



## Specifying the Keyboard Type



**Figure 1.** The default text input type.



**Figure 2.** The textEmailAddress input type.



**Figure 3.** The phone input type.

Text fields can have different input types, such as number, date, password, or email address. The type determines

what kind of characters are allowed inside the field, and may prompt the virtual keyboard to optimize its layout for

frequently used characters.

You can specify the type of keyboard you want for your[EditText](http://developer.android.com/reference/android/widget/EditText.html) object with the [android:inputType](http://developer.android.com/reference/android/widget/TextView.html#attr_android:inputType) attribute. For

example, if you want the user to input an email address, you should use the textEmailAddress input type:

<EditText

    android:id="@+id/email\_address"

    android:layout\_width="fill\_parent"

    android:layout\_height="wrap\_content"

    android:hint="@string/email\_hint"

    android:inputType="textEmailAddress" />

There are several different input types available for different situations. Here are some of the more common values

for[android:inputType](http://developer.android.com/reference/android/widget/TextView.html#attr_android:inputType):

"text"

Normal text keyboard.

"textEmailAddress"

Normal text keyboard with the @ character.

"textUri"

Normal text keyboard with the / character.

"number"

Basic number keypad.

"phone"

Phone-style keypad.

### Controlling other behaviors :

The [android:inputType](http://developer.android.com/reference/android/widget/TextView.html#attr_android:inputType) also allows you to specify certain keyboard behaviors, such as whether to capitalize all new

words or use features like auto-complete and spelling suggestions.

The [android:inputType](http://developer.android.com/reference/android/widget/TextView.html#attr_android:inputType) attribute allows bitwise combinations so you can specify both a keyboard layout and one or

more behaviors at once.

Here are some of the common input type values that define keyboard behaviors:

"textCapSentences" :

Normal text keyboard that capitalizes the first letter for each new

sentence.

"textCapWords" :

Normal text keyboard that capitalizes every word. Good for titles or person

names.

"textAutoCorrect" :

Normal text keyboard that corrects commonly misspelled words.

"textPassword" :

Normal text keyboard, but the characters entered turn into dots.

"textMultiLine" :

Normal text keyboard that allow users to input long strings of text that include line breaks (carriage returns).

For example, here's how you can collect a postal address, capitalize each word, and disable text suggestions:

<EditText

    android:id="@+id/postal\_address"

    android:layout\_width="fill\_parent"

    android:layout\_height="wrap\_content"

    android:hint="@string/postal\_address\_hint"

    android:inputType="textPostalAddress | textCapWords |  textNoSuggestions" />

All behaviors are also listed with the [android:inputType](http://developer.android.com/reference/android/widget/TextView.html#attr_android:inputType) documentation.

Android - MediaPlayer Tutorial

Android provides many ways to control playback of audio/video files and streams. One of this way is through a class called **MediaPlayer**.

Android is providing MediaPlayer class to access built-in mediaplayer services like playing audio,video e.t.c. In order to use MediaPlayer , we have to call a static Method **create()** of this class. This method returns an instance of MediaPlayer class. Its syntax is as follows −

MediaPlayer mediaPlayer = MediaPlayer.create(this, R.raw.song);

The second parameter is the name of the song that you want to play. You have to make a new folder under your project with name **raw** and place the music file into it.

Once you have created the Mediaplayer object you can call some methods to start or stop the music. These methods are listed below.

mediaPlayer.start();

mediaPlayer.pause();

On call to **start()** method, the music will start playing from the beginning. If this method is called again after the **pause()** method , the music would start playing from where it is left and not from the beginning.

In order to start music from the beginning , you have to call **reset()** method. Its syntax is given below.

mediaPlayer.reset();

Apart from the start and pause method, there are other methods provided by this class for better dealing with audio/video files. These methods are listed below −

|  |  |
| --- | --- |
| **Sr.No** | **Method & description** |
| 1 | **isPlaying()**  This method just returns true/false indicating the song is playing or not |
| 2 | **seekTo(position)**  This method takes an integer, and move song to that particular second |
| 3 | **getCurrentDuration()**  This method returns the current position of song in milliseconds |
| 4 | **getDuration()**  This method returns the total time duration of song in milliseconds |
| 5 | **reset()**  This method resets the media player |
| 6 | **release()**  This method releases any resource attached with MediaPlayer object |
| 7 | **setVolume(float leftVolume, float rightVolume)**  This method sets the up down volume for this player |
| 8 | **setDataSource(FileDescriptor fd)**  This method sets the data source of audio/video file |
| 9 | **selectTrack(int index)**  This method takes an integer, and select the track from the list on that particular index |
| 10 | **getTrackInfo()**  This method returns an array of track information |