History of Android

- The history and versions of android are interesting to know.
- The code names of android ranges from A to S currently, such as Aestro, Blender, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwitch, Jelly Bean, KitKat, Lollipop, Marshmallow upto Snow
- Let's understand the android history in a sequence.

1) Initially, Andy Rubin founded Android Incorporation in Palo Alto, California, United States in October, 2003.
2) In 17th August 2005, Google acquired android Incorporation. Since then, it is in the subsidiary of Google Incorporation.
3) The key employees of Android Incorporation are Andy Rubin, Rich Miner, Chris White and Nick Sears.
4) Originally intended for camera but shifted to smart phones later because of low market for camera only.
5) Android is the nick name of Andy Rubin given by coworkers because of his love to robots.
6) In 2007, Google announces the development of android OS.
7) In 2008, HTC launched the first android mobile.

Android Versions

Name	Version #	Release date	API
Android 1.0	1	Sept 23, 2008	1
Android 1.1	1.1	February 9, 2009	2
Android Cupcake	1.5	April 27, 2009	3
Android Donut	1.6	Sept 15, 2009	4
Android Eclair	2	October 27, 2009	5
	2.0.1	December 3, 2009	6
	2.1	January 11, 2010	7
Android Froyo	2.2 – 2.2.3	May 20, 2010	8
Android Gingerbread	2.3 – 2.3.2	December 6, 2010	9
	2.3.3 - 2.3.7	February 9, 2011	10
Android Honeycomb	3	February 22, 2011	11
	3.1	May 10, 2011	12
	3.2 - 3.2.6	July 15, 2011	13
Android Ice Cream Sandwich	4.0 – 4.0.2	October 18, 2011	14
	4.0.3 - 4.0.4	December 16, 2011	15
Android Jelly Bean	4.1 – 4.1.2	July 9, 2012	16
	4.2 - 4.2.2	November 13, 2012	17
	4.3 - 4.3.1	July 24, 2013	18
Android KitKat	4.4 – 4.4.4	October 31, 2013	19
	4.4W - 4.4W.2	June 25, 2014	20
Android Lollipop	5.0 – 5.0.2	November 4, 2014	21
	5.1 - 5.1.1	March 2, 2015	22
Android Marshmallow	6.0 – 6.0.1	October 2, 2015	23
Android Nougat	7	August 22, 2016	24
	7.1 - 7.1.2	October 4, 2016	25
Android Oreo	8	August 21, 2017	26
	8.1	December 5, 2017	27
Android Pie	9	August 6, 2018	28
Android 10	10	Sept 3, 2019	29
Android 11	11	Sept 8, 2020	30
Android 12	12	May 18, 2021	31
Android 12L	12.1	March 7, 2022	32
Android 13	13	Q3 2022	33

ANDROID DEVELOPMENT TOOLS

The Android Virtual Device and SDK Managers

- Used to create and manage AVDs and to download SDK packages, respectively.
- The AVD hosts an Emulator running a particular build of Android, letting you specify the supported SDK version, screen resolution, amount of SD card storage available, and available hardware capabilities (such as touchscreens and GPS).

The Android Emulator

- An implementation of the Android VM designed to run within an AVD on your development computer.
- Use the Emulator to test and debug your Android applications.

Dalvik Debug Monitoring Service (DDMS)

- Use the DDMS to monitor and control the Emulators on which you're debugging your applications

Android Debug Bridge (ADB)

- A client-server application that provides a link to virtual and physical devices.
- It lets you copy files, install compiled application packages (.apk), and run shell commands.

Logcat

- A utility used to view and filter the output of the Android logging system.

Android Asset Packaging Tool (AAPT)

- Constructs the distributable Android package files (.apk).

The following additional tools are also available:

- **SQLite3** — A database tool that you can use to access the SQLite database fi les created and used by Android.
- **Traceview and dmtracedump** — Graphical analysis tools for viewing the trace logs from your Android application.
- **Hprof-conv** — A tool that converts HPROF profi ling output fi les into a standard format to view in your preferred profiling tool.
- **MkSDCard** — Creates an SD card disk image that can be used by the Emulator to simulate an external storage card.
- **Dx** — Converts Java .class bytecode into Android .dex bytecode.
- **Hierarchy Viewer** — Provides both a visual representation of a layout's View hierarchy to debug and optimize your UI, and a magnified display to get your layouts pixel-perfect.
- **Lint** — A tool that analyzes your application and its resources to suggest improvements and optimizations.

Basics of Android Application

The following components comprise the building blocks for all your Android applications:

Activities

- Your application's presentation layer.
- The UI of your application is built around one or more extensions of the Activity class.
- Activities use Fragments and Views to layout and display information, and to respond to user actions. Compared to desktop development, Activities are equivalent to Forms.

Services

- The invisible workers of your application. Service components run without a UI, updating your data sources and Activities, triggering Notifications, and broadcasting Intents.
- They're used to perform long running tasks, or those that require no user interaction (such as network lookups or tasks that need to continue even when your application's Activities aren't active or visible.)

Content Providers

- Shareable persistent data storage.
- Content Providers manage and persist application data and typically interact with SQL databases.
- They're also the preferred means to share data across application boundaries. You can configure your application's Content Providers to allow access from other applications, and you can access the Content Providers exposed by others.
- Android devices include several native Content Providers that expose useful databases such as the media store and contacts.

Intents

- A powerful inter application message-passing framework.
- Intents are used extensively throughout Android.
- You can use Intents to start and stop Activities and Services, to broadcast messages system-wide or to an explicit Activity, Service, or Broadcast Receiver, or to request an action be performed on a particular piece of data

Broadcast Receivers

- Intent listeners.
- Broadcast Receivers enable your application to listen for Intents that match the criteria you specify.
- Broadcast Receivers start your application to react to any received Intent, making them perfect for creating event-driven applications.

Widgets

- Visual application components that are typically added to the device home screen.
- A special variation of a Broadcast Receiver, widgets enable you to create dynamic, interactive application components for users to embed on their home screens.

Notifications

- Notifications enable you to alert users to application events without stealing focus or interrupting their current Activity.
- They're the preferred technique for getting a user's attention when your application is not visible or active, particularly from within a Service or Broadcast Receiver.
- For example, when a device receives a text message or an email, the messaging and Gmail applications use Notifications to alert you by flashing lights, playing sounds, displaying icons, and scrolling a text summary

Android Manifest File

- The **AndroidManifest.xml file** *contains information of your package*, including components of the application such as activities, services, broadcast receivers, content providers etc.

It performs some other tasks also:

- It is **responsible to protect the application** to access any protected parts by providing the permissions.
- It also **declares the android api** that the application is going to use.
- It **lists the instrumentation classes**.
- The instrumentation classes provides profiling and other information.
- This information is removed just before the application is published etc.

A Closer Look at the Application Manifest The following XML snippet shows a typical manifest node:

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
  package="avt.mca.atmiya.testapp">

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"

android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/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>
```

Elements of the AndroidManifest.xml file

The elements used in the above xml file are described below.

<manifest>

manifest is the root element of the AndroidManifest.xml file. It has package attribute that describes the package name of the activity class.

<application>

- application is the sub element of the manifest.
- It includes the namespace declaration.
- This element contains several sub elements that declares the application component such as activity etc.
- The commonly used attributes are of this element are icon, label, theme etc.
- **android:icon** represents the icon for all the android application components.
- **android:label** works as the default label for all the application components.
- **android:theme** represents a common theme for all the android activities.

<activity>

- activity is the sub element of application and represents an activity that must be defined in the AndroidManifest.xml file.
- It has many attributes such as label, name, theme, launch Mode etc.
- **android:label** represents a label i.e. displayed on the screen.
- **android:name** represents a name for the activity class. It is required attribute.

<intent-filter>

intent-filter is the sub-element of activity that describes the type of intent to which activity, service or broadcast receiver can respond to.

<action>

It adds an action for the intent-filter. The intent-filter must have at least one action element.

<category>

It adds a category name to an intent-filter.

<uses-permissions>

used to specify permissions that are requested for the purpose of security.

Resource in Android

String
- XML resource that carries a string value.

Bool
- XML resource that carries a boolean value.

Color
- XML resource that carries a color value (a hexadecimal color).

Dimension
- XML resource that carries a dimension value (with a unit of measure).

ID
- XML resource that provides a unique identifier for application resources and components.

Integer
- XML resource that carries an integer value.

Integer Array
- XML resource that provides an array of integers.

A Simple Example

```
<resources>
<string name=”app_name”>To Do List</string>
<color name=”app_background”>#000</color>
<dimen name=”default_border”>5px</dimen>
<string-array name=“string_array“>
        <item>Item 1</item>
        <item>Item 2</item>
       <item>Item 3</item>
 </string-array>
<array name=“integer_array“>
        <item>3</item>
        <item>2</item>
        <item>1</item>
  </array>
</resources>
```

Bool

A boolean value defined in XML.

A bool is a simple resource that is referenced using the value provided in the name attribute.

Syntax

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <bool name="bool_name"> [true | false]</bool>
</resources>
```

EXAMPLE:

XML file saved at res/values-small/bools.xml:

```
<resources>
   <bool name="screen_small">true</bool>
   <bool name="adjust_view_bounds">true</bool>
</resources>
```

This application code retrieves the boolean:

```
var b:boolean
=resources.getBoolean(R.id.screen_small)
```

Color Resource

- A color value defined in XML.
- The color is specified with an RGB value and alpha channel.
- You can use a color resource any place that accepts a hexadecimal color value.
- You can also use a color resource when a drawable resource is expected in XML (for example, android:drawable="@color/green".
- The value always begins with a pound (#) character and then followed by the Alpha-Red-Green-Blue information in one of the following formats:
  #RGB
  #ARGB
  #RRGGBB
  #AARRGGBB

Syntax

```
<resources>
  <color name="color_name">hex_color</color>
</resources>
```

EXAMPLE:

XML file saved at res/values/colors.xml:

```
<resources>
  <color name="opaque_red">#f00</color>
</resources>
```

This application code retrieves the color resource:

```
var res = resources
var color = res.getColor(R.color.opaque_red);
```

Dimension

- A dimension value defined in XML.
- A dimension is specified with a number followed by a unit of measure.
- For example: 10px, 2in, 5sp.

The following units of measure are supported by Android:

dp

- Density-independent Pixels –
- An abstract unit that is based on the physical density of the screen.
- These units are relative to a 160 dpi (dots per inch) screen, on which 1dp is roughly equal to 1px.

sp

- 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 the user's preference.

pt

- Points - 1/72 of an inch based on the physical size of the screen, assuming a 72dpi density screen.

px

- Pixels - Corresponds to actual pixels on the screen.
- This unit of measure is not recommended because the actual representation can vary across devices; each devices may have a different number of pixels per inch and may have more or fewer total pixels available on the screen.

in

Inches - Based on the physical size of the screen.

Example

XML file saved at `res/values/dimens.xml`:

```
<resources>
   <dimen name="textview_height">25dp</dimen>
   <dimen name="textview_width">150dp</dimen>
   <dimen name="font_size">16sp</dimen>
</resources>
```

```
var res = resources
var f = res.getDimension(R.dimen.font_size)
```

Integer

An integer defined in XML

Example

XML file saved at res/values/integers.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <integer name="max_speed">75</integer>
   <integer name="min_speed">5</integer>
</resources>
```

```
Resources res = getResources();
int speed = res.getInteger(R.integer.max_speed);
```

Integer Array Resource

- An array of integers defined in XML.

EXAMPLE:

XML file saved at res/values/integers.xml:

```
<resources>
   <integer-array name="bits">
      <item>4</item>
      <item>8</item>
      <item>16</item>
      <item>32</item>
   </integer-array>
</resources>
```

```
var res = resources
var bits = res.getIntArray(R.array.bits);
```

ID Resource

- A unique resource ID defined in XML.

EXAMPLE:

XML file saved at res/values/ids.xml:

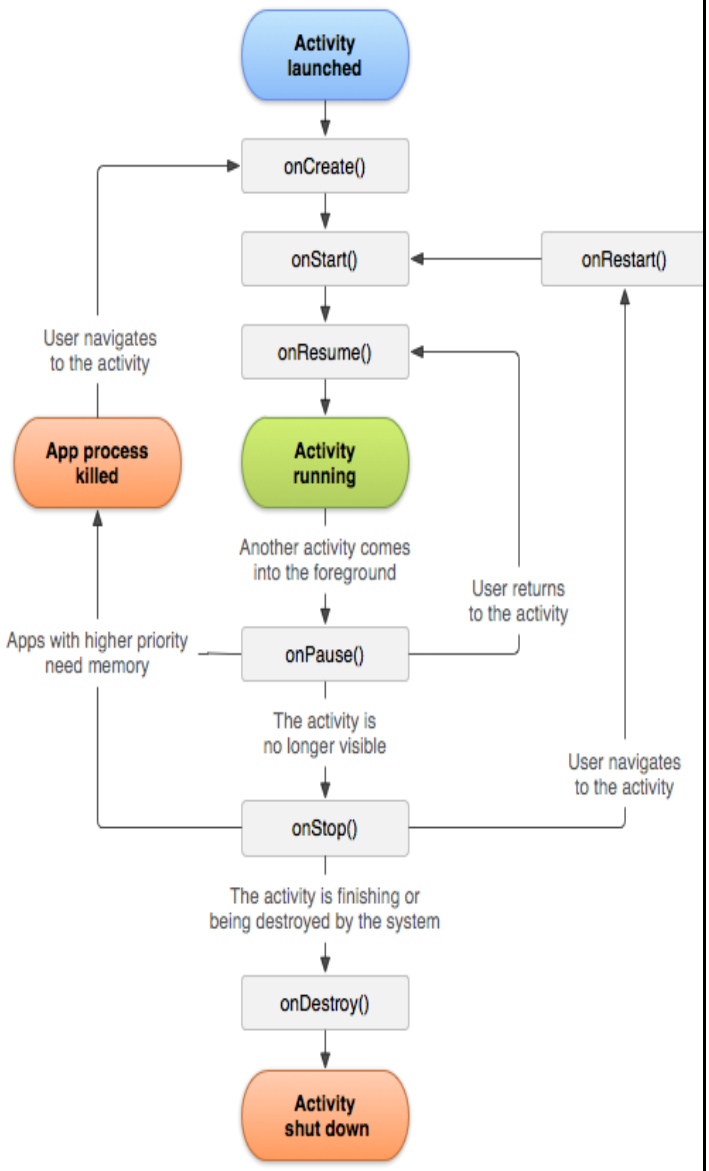
```
<resources>
   <item type="id" name="button_ok" />
</resources>
```

```
<Button
android:id="@id/button_ok"
style="@style/button_style" />
```

- Notice that the android:id value does not include the plus sign in the ID reference, because the ID already exists, as defined in the ids.xml example above.

Explain the lifecycle of the Android activity.

- "An activity represents a single screen with a user interface just like window or frame of Java.
- Android activity is the subclass of ContextThemeWrapper class."
- Android system initiates its program within an Activity starting with a call on onCreate() call-back method, similarly Like Main() Function.



Entire Lifetime: This is the lifetime between the first call to the onCreate() and the final call to onDestroy()method.

Visible Lifetime: It is the lifetime of an Activity between onStart() and onStop() method calls.

Call back With Its Description

onCreate()

- This is the first callback and called when the activity is first created.
- This is where you should do all of your normal static set up: create views, bind data to lists, etc.

onStart()

- This callback is called when the activity becomes visible to the user.
- onStart() method is called before the Activity is being visible to the User. Activity is still not Active.

onResume()

- This is called when the user starts interacting with the application.

onPause()

- The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.

onStop()

This callback is called when the activity is no longer visible.

onDestroy()

This callback is called before the activity is destroyed by the system.

onRestart()

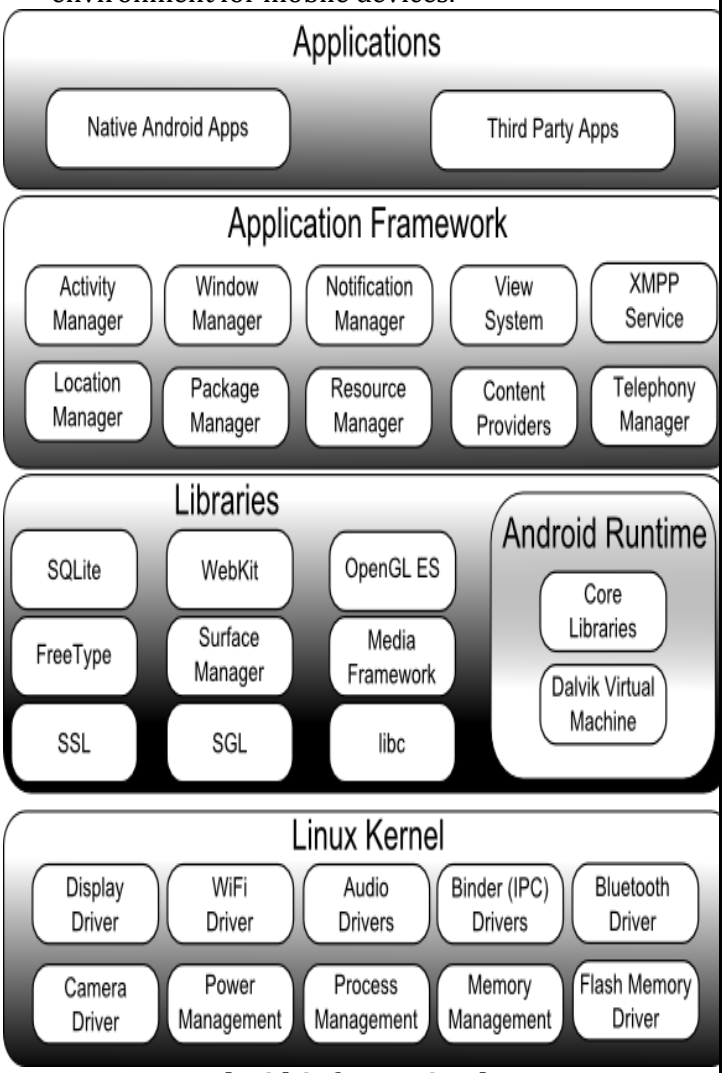
This callback is called when the activity restarts after stopping it.

Example

```
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.util.Log

class MainActivity : AppCompatActivity() {

   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
      print("onCreate")
   }

   override fun onStart() {
      super.onStart()
      print("onStart")
   }

   override fun onResume() {
      super.onResume()
      print("onResume")
   }

   override fun onPause() {
      super.onPause()
      print("onPause")
   }

   override fun onStop() {
      super.onStop()
      print("onStop")
   }

   override fun onRestart() {
      super.onRestart()
      print("onRestart")
   }

   override fun onDestroy() {
      super.onDestroy()
      print("onDestroy")
   }

   fun print(msg: String){
      Toast.makeText(applicationContext, msg,
Toast.LENGTH_LONG).show()
   }
}
```

Explain Android Software Stack
OR
Explain Android Architecture

- Android is architected in the form of a software stack comprising applications, an operating system, run-time environment, middleware, services and libraries.
- This architecture can, perhaps, best be represented visually as outlined in Following Diagram. E
- ach layer of the stack, and the corresponding elements within each layer, are tightly integrated and carefully tuned to provide the optimal application development and execution environment for mobile devices.



Android Software Stack

The Linux Kernel

- Positioned at the bottom of the Android software stack, the Linux Kernel provides a level of abstraction between the device hardware and the upper layers of the Android software stack.
- Based on Linux version 2.6, the kernel provides preemptive multitasking, low-level core system services such as memory, process and power management in addition to providing a network stack and device drivers for hardware such as the device display, Wi-Fi and audio.

Android Runtime - Dalvik Virtual Machine

- As previously noted, the Linux kernel provides a multitasking execution environment allowing multiple processes to execute concurrently.
- It would be easy to assume, therefore, that each Android application simply runs as a process directly on the Linux kernel.
- In fact, each application running on an Android device does so within its own instance of the Dalvik virtual machine (VM).
- The Dalvik virtual machine was developed by Google and relies on the underlying Linux kernel for low-level functionality.
- It is more efficient than the standard Java VM in terms of memory usage, and specifically designed to allow multiple instances to run efficiently within the resource constraints of a mobile device.

Android Libraries

A summary of some key core Android libraries:

- **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.database** – Used to access data published by content providers and includes SQLite database management classes.
- **android.graphics** – A low-level 2D graphics drawing API including colors, points, filters, rectangles and canvases.
- **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.

Application Framework

- The Application Framework is a set of services that collectively form the environment in which Android applications run and are managed.
- This framework implements the concept that Android applications are constructed from reusable, interchangeable and replaceable components.

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.
- **Package Manager**
  - The system by which applications are able to find out information about other applications currently installed on the device.
- **Telephony Manager**
  - Provides information to the application about the telephony services available on the device such as status and subscriber information.
- **Location Manager**
  - Provides access to the location services allowing an application to receive updates about location changes.

Applications

- Located at the top of the Android software stack are the applications.
- These comprise both the native applications provided with the particular Android implementation (for example web browser and email applications) and the third party applications installed by the user after purchasing the device.

Summary

- A good Android development knowledge foundation requires an understanding of the overall architecture of Android.
- Android is implemented in the form of a software stack architecture consisting of a Linux kernel, a runtime environment and corresponding libraries, an application framework and a set of applications. Applications are predominantly written in Java and run within individual instances of the Dalvik virtual machine.
- The key goals of the Android architecture are performance and efficiency, both in application execution and in the implementation of reuse in application design.

What is Layout? List various layouts and explain any two in detail

- Layouts are used to organize screen elements (user interface)
- Layouts can be created in → xml or in java programmatically.
- Layouts can be created in xml file in a → res/layout directory.
- To set layout designed in xml on screen →setContentView(R.layout.test) is used.
- Built in layout in Android SDK includes
  - FrameLayout
  - LinearLayout
  - RelativeLayout
  - TableLayout

Using FrameLayout

- A FrameLayout view is designed to display a stack of child View items.
- You can add multiple views to this layout, but each View is drawn from the top-left corner of the layout.
- You can use this to show multiple images within the same region.

Example of Frame Layout

- Here's an example of an XML layout resource with a FrameLayout and two child View objects, both ImageView objects.
- The green rectangle is drawn first and the red oval is drawn on top of it.
- The green rectangle is larger, so it defines the bounds of the FrameLayout:

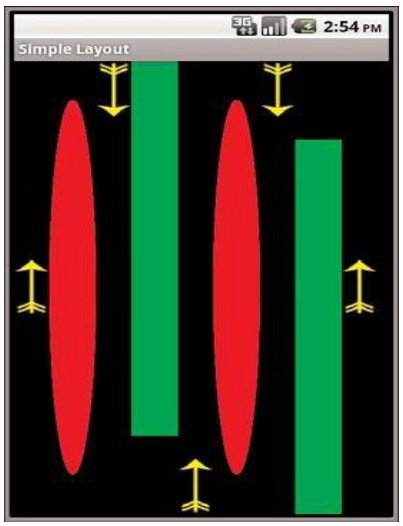
```
<FrameLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
android:id="@+id/FrameLayout01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center">

<ImageView
android:id="@+id/ImageView01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/green_rect"
android:minHeight="200px"
android:minWidth="200px" />

<ImageView
android:id="@+id/ImageView02"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/red_oval"
android:minHeight="100px"
android:minWidth="100px"
android:layout_gravity="center" />
</FrameLayout>
```

FrameLayout Attributes

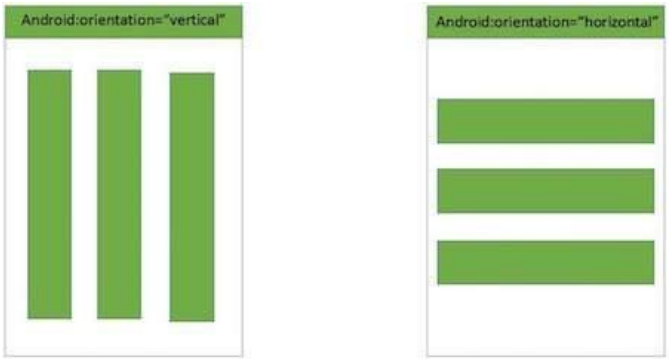
Attribute	Description
android:id	This is the ID which uniquely identifies the layout.
android:foreground	This defines the drawable to draw over the content and possible values may be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
android:foregroundGravity	Defines the gravity to apply to the foreground drawable. Possible values are top, bottom,left,right,center

Using LinearLayout

- A LinearLayout view organizes its child View objects in a single row, shown in following diagram, or column, depending on whether its orientation attribute is set to horizontal or vertical.
- This is a very handy layout method for creating forms.



LinearLayout : Horizontal



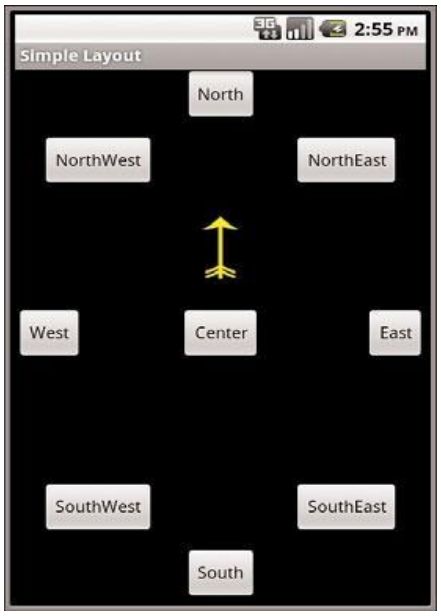
LinearLayout is a view group that aligns all children in either *vertically* or *horizontally*.

Following are the important attributes specific to LinearLayout –

Attribute	Description
android:id	This is the ID which uniquely identifies the layout.
android:divider	This is drawable to use as a vertical divider between buttons. You use a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
android:gravity	This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
android:orientation	This specifies the direction of arrangement and you will use "horizontal" for a row, "vertical" for a column. The default is horizontal.

Using RelativeLayout

- The RelativeLayout view enables you to specify where the child view controls are in relation to each other.
- For instance, you can set a child View to be positioned “above” or “below” or “to the left of ” or “to the right of ” another View, referred to by its unique identifier.
- You can also align child View objects relative to one another or the parent layout edges.
- Combining RelativeLayout attributes can simplify creating interesting user interfaces without resorting to multiple layout groups to achieve a desired effect.
- Following Diagram shows how each of the button controls is relative to each other.



Relative Layout

Important attributes specific to RelativeLayout

Attribute	Description
android:id	This is the ID which uniquely identifies the layout.
android:gravity	This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
android:ignoreGravity	This indicates what view should not be affected by gravity.

Layout properties available from **RelativeLayout.LayoutParams** and few of the important attributes are given below

Attribute	Description
android:layout_above	Positions the bottom edge of this view above the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name"
android:layout_alignBottom	Makes the bottom edge of this view match the bottom edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_alignParentBottom	If true, makes the bottom edge of this view match the bottom edge of the parent. Must be a boolean value, either "true" or "false".

android:layout_alignParentRight	If true, makes the right edge of this view match the right edge of the parent. Must be a boolean value, either "true" or "false".
android:layout_alignParentStart	If true, makes the start edge of this view match the start edge of the parent. Must be a boolean value, either "true" or "false".
android:layout_alignParentTop	If true, makes the top edge of this view match the top edge of the parent. Must be a boolean value, either "true" or "false".
android:layout_alignRight	Makes the right edge of this view match the right edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_alignStart	Makes the start edge of this view match the start edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_alignTop	Makes the top edge of this view match the top edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_below	Positions the top edge of this view below the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_centerHorizontal	If true, centers this child horizontally within its parent. Must be a boolean value, either "true" or "false".
android:layout_centerInParent	If true, centers this child horizontally and vertically within its parent. Must be a boolean value, either "true" or "false".
android:layout_centerVertical	If true, centers this child vertically within its parent. Must be a boolean value, either "true" or "false".
android:layout_toEndOf	Positions the start edge of this view to the end of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_toLeftOf	Positions the right edge of this view to the left of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".

Example

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android=
“http://schemas.android.com/apk/res/android"
android:id="@+id/RelativeLayout01"
android:layout_height="fill_parent"
android:layout_width="fill_parent">

<Button
android:id="@+id/ButtonCenter"
android:text="Center"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerInParent="true" />

<ImageView
android:id="@+id/ImageView01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_above="@id/ButtonCenter"
android:layout_centerHorizontal="true"
android:src="@drawable/arrow" />
</RelativeLayout>
```

TableLayout

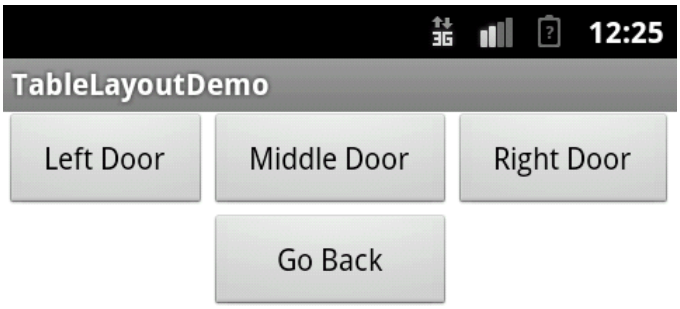
- A TableLayout view organizes children into rows, as shown in Example Diagram.
- You add individual View objects within each row of the table using a TableRow layout View (which is basically a horizontally oriented LinearLayout) for each row of the table.
- Each column of the TableRow can contain one View (or layout with child View objects).You place View items added to a TableRow in columns in the order they are added.
- You can specify the column number (zero-based) to skip columns as necessary, otherwise, the View object is put in the next column to the right.
- Columns scale to the size of the largest View of that column.
- You can also include normal View objects instead of TableRow elements, if you want the View to take up an entire row.

Following are the important attributes specific to TableLayout –

Attribute	Description
android:collapseColumns	This specifies the zero-based index of the columns to collapse. The column indices must be separated by a comma: 1, 2, 5.
android:layout_column	We can provide column no to the view.
android:stretchColumns	The zero-based index of the columns to stretch. The column indices must be separated by a comma: 1, 2, 5.

Example

```
<TableLayout xmlns:android=
   "http://schemas.android.com/apk/res/android"
   android:id="@+id/TableLayout01"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:stretchColumns="*">
   <TableRow
      android:id="@+id/TableRow01">
      <Button
         android:id="@+id/ButtonLeft"
         android:text="Left Door" />
      <Button
         android:id="@+id/ButtonMiddle"
         android:text="Middle Door" />
      <Button
         android:id="@+id/ButtonRight"
         android:text="Right Door" />
   </TableRow>
   <TableRow
      android:id="@+id/TableRow02">
      <Button
         android:id="@+id/ButtonBack"
         android:text="Go Back"
         android:layout_column="1" />
   </TableRow>
</TableLayout>
```



Using Multiple Layouts on a Screen

- Combining different layout methods on a single screen can create complex layouts.
- Remember that because a layout contains View objects and is, itself, a View, it can contain other layouts.
- Following diagram demonstrates a combination of layout views used in conjunction to create a more complex and interesting screen.

What is Intent?

- Android application components can connect to other Android applications.
- This connection is based on a task description represented by an Intent object.

Android intents are mainly used to:

- Start the service
- Launch an activity
- Display a web page
- Display a list of contacts
- Broadcast a message
- Dial a phone call etc.

- *Intents* are asynchronous messages which allow application components to request functionality from other Android components.
- Intents allow you to interact with components from the same applications as well as with components contributed by other applications.
- For example, an activity can start an external activity for taking a picture.
- Intents are objects of the android.content.Intent type.
- Your code can send them to the Android system defining the components you are targeting.
- For example, via the startActivity() method you can define that the intent should be used to start an activity.
- An intent can contain data via a Bundle. This data can be used by the receiving component.

The following code demonstrates how you can start another activity via an intent.

```
Intent i = new Intent(this, ActivityTwo.class);
startActivity(i);
```

Sending out explicit or implicit intents

- Android supports explicit and implicit intents.
- An application can define the target component directly in the intent (*explicit intent*) or ask the Android system to evaluate registered components based on the intent data (*implicit intents*).

- **Explicit intents** explicitly define the component which should be called by the Android system, by using the Java class as identifier. Explicit intents are typically used within an application as the classes in an application are controlled by the application developer.

Explicit Intent Example

To Pass Value and start ActivityTwo

```
var i = Intent(this,ActivityTwo::class.java)
i.putExtra("Value1", "Value1 for ActivityTwo ");
i.putExtra("Value2", "Value2 for ActivityTwo");
startActivity(i)
```

To get Value in ActivityTwo

```
var val1 = intent.getStringExtra("Value1")
Toast.makeText(applicationContext, val1 ,
Toast.LENGTH_LONG).show()
```

Implicit Intent Example

To Open Website URL

```
var uri = Uri.parse("https://google.com")
var i = Intent(Intent.ACTION_VIEW, uri)
startActivity(i)
```

To Dial a Number

```
var uri = Uri.parse("tel:"+editText3.text.toString())
var i = Intent(Intent.ACTION_DIAL, uri)
startActivity(i)
```

To Open Location in Map

```
var uri = Uri.parse("geo:0,0?q="+"Atmiya
University, Rajkot")
var i = Intent(Intent.ACTION_VIEW, uri)
startActivity(i)
```

To Share Text to User

```
ShareCompat.IntentBuilder.from(this)
  .setChooserTitle("Share Text")
  .setText(editText5.text.toString())
  .setType("plain/text")
  .startChooser()
```

- Using implicit intent we can send sms, mms, capture image, capture video, dial phone, call phone etc..

Intents as event triggers

- Intents can be used to send broadcast messages into the Android system.
- A *broadcast receiver* can register to an event and is notified if such an event is sent.
- Your application can register to system events, e.g., a new email has arrived, system boot is complete or a phone call is received and react accordingly.

Registering intent via intent filters

Register an activity as a launcher

```
<activity android:name=".ViewFlipperActivity">
  <intent-filter>
    <action android:name=
          "android.intent.action.MAIN" />
    <category android:name=
          "android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

Intent : Data transfer to the target component

- An intent contains certain header data, e.g., the desired action, the type, etc.
- Optionally an intent can also contain additional data based on an instance of the Bundle class which can be retrieved from the intent via the getExtras() method.
- You can also add data directly to the Bundle via the overloaded putExtra() methods of the Intent objects.

```
var i = Intent(this,WidgetActivity::class.java)
i.putExtra("Value1","Value1 for Activity Two")
i.putExtra("Value2","Value2 for Activity Two")
startActivity(i)

var extras = getIntent().getExtras();
if (extras == null) {
   return;
}
// get data via the key
var value1 = extras.getString("Value1");
if (value1 != null) {
   // do something with the data
}
```

Codes

Button Code

```
button.setOnClickListener { it: View!
    Toast.makeText(applicationContext,
        text: "Atmiya University...!",
        Toast.LENGTH_LONG).show()
}
```

Toggle Button

```
toggleButton.setOnClickListener{ it: View!
    if(toggleButton.text == "બંધ")
        imageView.setImageResource(R.drawable.bulboff)
    else
        imageView.setImageResource(R.drawable.bulbon)
}
```

Radio Button

```
radioGroup.setOnCheckedChangeListener { rgroup, checkedId ->
    var rb: RadioButton? = findViewById(checkedId)
    if(rb!=null)
        textView4.setText(rb.text)
}
```

To Clear All Option Selected in Radio Group

```
radioGroup.clearCheck()
textView4.setText("Choose Option")
```

CheckBox Code

```
checkBox.setOnClickListener { it: View!
    if(checkBox.isChecked)
        android = "YES"
    else
        android = "NO"
    textView2.setText("Android = $android")
}
```

DatePicker & TimePicker Dialog

```
var c = Calendar.getInstance()
DatePickerDialog( context: this, DatePickerDialog.OnDateSetListener
        { datePicker, i, i2, i3 ->
            var dt = "$i3/${i2+1}/$i"
            TimePickerDialog( context: this,
                TimePickerDialog.OnTimeSetListener { timePicker, i, i2 ->
                        dt+=" $i:$i2"
                        editText5.setText("$dt")
                },c.get(Calendar.HOUR),c.get(Calendar.MINUTE), is24HourView: false).show()
},c.get(Calendar.YEAR),c.get(Calendar.MONTH),c.get(Calendar.DAY_OF_MONTH)).show()
```

Alert Dialog

```
AlertDialog.Builder( context: this).setTitle("Enjoying...!")
    .setMessage("Are You Enjoying???")
    .setPositiveButton( text: "Yes",
        DialogInterface.OnClickListener { dialogInterface, i ->
                Toast.makeText(applicationContext, text: "Thank God..!",Toast.LENGTH_LONG).show()
    })
    .setNegativeButton( text: "No", DialogInterface.OnClickListener { dialogInterface, i ->
        Toast.makeText(applicationContext, text: "Oh God\nI will teach you again\nI apologize..!",
            Toast.LENGTH_LONG).show()
    })
    .show()
```

Progress Dialog

```
var p = ProgressDialog( context: this)
p.setTitle("Downloading")
p.setMessage("File Downloading")
p.max = 100
p.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL)
p.show()

var count = 0
Thread(Runnable {
    while(count<=100){
        try{
            p.progress = count
            count += 10
            Thread.sleep( millis: 1000)
        }catch (e:InterruptedException ){}
    }
}).start()
```

Implicit Intent

```
fun openwebsite(view: View) {
    var uri = Uri.parse(editText.text.toString())
    startActivity(Intent(Intent.ACTION_VIEW,uri))
}
fun openDialPad(view: View) {
    var uri = Uri.parse( uriString: "tel:"+editText2.text.toString())
    startActivity(Intent(Intent.ACTION_DIAL,uri))
}
fun openMap(view: View) {
    var uri = Uri.parse( uriString: "geo:0,0?q="+editText3.text.toString())
    startActivity(Intent(Intent.ACTION_VIEW,uri))
}
fun shareText(view: View) {
    ShareCompat.IntentBuilder
        .from( launchingActivity: this)
        .setText(editText4.text.toString())
        .setType("text/plain")
        .startChooser()
}
```

ListView

```
var arr : Array<String> = Array( size: 5){""}
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    arr = listOf<String>("Program-1","Program-2","Program-3","Program-4","Program-5")
                    .toTypedArray()
    var adapter = ArrayAdapter( context: this,
        //android.R.layout.simple_expandable_list_item_1,
        R.layout.mylayout,
        arr)
    listview.setAdapter(adapter)

    listview.setOnItemClickListener { adapterView, view, pos, myfun ->
        when(pos){
            0-> {startActivity(Intent(applicationContext,Program1::class.java))}
            1-> {startActivity(Intent(applicationContext,Program2::class.java))}
        }
    }

}
```

Splash Screen Code

```
Handler().postDelayed({
    var i = Intent(applicationContext,MainActivity::class.java)
    startActivity(i)
    finish()
}, delayMillis: 4000)
```

Edit Text – Some One Typing Checking

```
editText.addTextChangedListener(object:TextWatcher{
    override fun afterTextChanged(p0: Editable?) {
    }
    override fun beforeTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {
    }
    override fun onTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {
        textView2.setText(p0)
        textView3.visibility = View.VISIBLE
        var t:Timer = Timer()
        t.schedule(object:TimerTask(){
            override fun run() {
                textView3.visibility = View.INVISIBLE
            }
        }, delay: 2000)
    }
})
```

Edit Text - Show Error if Email Address is Invalid

```
editText.addTextChangedListener(object:TextWatcher{
    override fun afterTextChanged(p0: Editable?) {
    }

    override fun beforeTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {
    }

    override fun onTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {
        if(!android.util.Patterns.EMAIL_ADDRESS.matcher(editText.text).matches()){
            editText.setError("Please enter valid Email")
        }
    }
})
```

Show Password – Material Design

```
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/textInputLayout2"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:endIconMode="password_toggle"
    app:startIconDrawable="@drawable/ic_baseline_lock_24">

    <com.google.android.material.textfield.TextInputEditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Password"
        android:inputType="textPassword" />
</com.google.android.material.textfield.TextInputLayout>
```

Common Android Views

TextView

Displays text

```
<TextView
    android:id="@+id/title_text_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/my_photos"
    android:textAppearance="?android:textAppearanceLarge"
    android:textColor="#4689C8"
    android:textStyle="bold" />
```

My Photos

ImageView

Displays Image

```
<ImageView
    android:id="@+id/photo_image_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:scaleType="centerCrop"
    android:src="@drawable/beach" />
```

Button

Button with text label

```
<Button
    android:id="@+id/next_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/next" />
```

NEXT

View

Plain rectangle (can be used as a divider)

```
<View
    android:layout_width="match_parent"
    android:layout_height="100dp"
    android:background="#4E4B4F" />
```

EditText

Text field that you can type into

```
<EditText
    android:id="@+id/album_description_view"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="@string/album_description"
    android:inputType="textMultiLine" />
```

Album Desciption

Spinner

Click on it to show a list of dropdown options

```
<Spinner
    android:id="@+id/sort_by_spinner"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

Create SpinnerAdapter in Java code to populate the options. See more

Beach

Beach
BBQ
Family dinner
Party

CheckBox

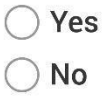
Checkbox with text label

```
<CheckBox
    android:id="@+id/notify_me_checkbox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/notify_me"
    android:textAppearance="?android:textAppearanceMedium" />
```

RadioButton

Radio button (where you can select one out of a group of radio buttons)

```
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton
        android:id="@+id/yes_radio_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/yes"
        android:textAppearance="?android:textAppearanceMedium" />
    <RadioButton
        android:id="@+id/no_radio_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/no"
        android:textAppearance="?android:textAppearanceMedium" />
</RadioGroup>
```



RatingBar

Star rating

```
<RatingBar
    android:id="@+id/rating_bar"
    style="?android:attr/ratingBarStyleSmall"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:numStars="5"
    android:rating="2.5"
    android:stepSize="0.5" />
```

Switch

On / off switch that you can drag right or left (or just tap to toggle the state)

```
<Switch
    android:id="@+id/backup_photos_switch"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/auto_backup_photos"
    android:textAppearance="?android:textAppearanceSmall" />
```

SeekBar

Displays progress and allows you to drag the handle anywhere in the bar (i.e. for music or video player)

```
<SeekBar
    android:id="@+id/seek_bar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:max="100"
    android:progress="20" />
```

SearchView

A search field that you can type a query into

```
<SearchView
    android:id="@+id/search_viewr"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:iconifiedByDefault="false"
    android:queryHint="@string/search_photos" />
```

ProgressBar

Loading spinner

```
<ProgressBar
    android:id="@+id/loading_spinner"
    style="?android:progressBarStyle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

ProgressBar

Horizontal loading indicator

```
<ProgressBar
    android:id="@+id/progress_bar"
    style="?android:progressBarStyleHorizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:indeterminate="false"
    android:max="100"
    android:progress="40"/>
```


Note: This is based on an app with minimum SDK version of Ice Cream Sandwich (API 15), where the activity theme in res/styles.xml is defined as the following. You can define other theme colors like colorPrimary in your app, see this guide.

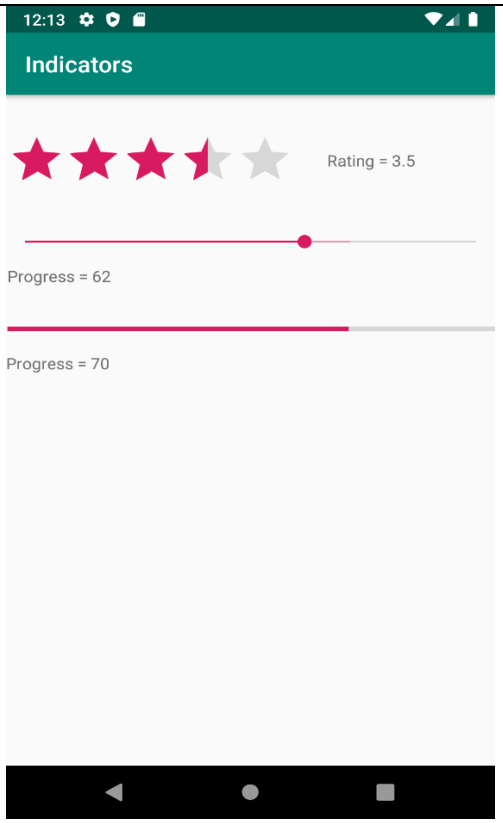
```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <item name="colorAccent">#4689C8</item>
</style>
```

INDICATORS – RatingBar, SeekBar & ProgressBar

```
<RatingBar
   android:id="@+id/ratingBar"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:numStars="5"
   android:stepSize=".25"/>

<TextView
   android:id="@+id/textView6"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="TextView" />
```

```
ratingBar.onRatingBarChangeListener =
   RatingBar.OnRatingBarChangeListener { ratingBar, rating, b ->
      textView6.text = "Rating = $rating"
   }
```

```
<SeekBar
   android:id="@+id/seekBar"
   android:layout_width="match_parent"
   android:layout_height="wrap_content" />

<TextView
   android:id="@+id/textView7"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="TextView" />

<ProgressBar
   android:id="@+id/myprogressBar"
   style="?android:attr/progressBarStyleHorizontal"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"/>

<TextView
   android:id="@+id/textView8"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="TextView" />
```



SeekBar

```
seekBar.setOnSeekBarChangeListener(object : SeekBar.OnSeekBarChangeListener{
    override fun onProgressChanged(p0: SeekBar?, p1: Int, p2: Boolean) {
        textView7.setText("Progress = $p1")
        seekBar.secondaryProgress = p1 + 10
    }
    override fun onStartTrackingTouch(p0: SeekBar?) {
    }
    override fun onStopTrackingTouch(p0: SeekBar?) {
    }
})
```

ProgressBar

```
Thread(Runnable {
    var count = 0
    while(count<=100){
        try {
            Thread.sleep( millis: 1000)
            count += 10
            myprogressBar.progress = count
        }catch (e:InterruptedException){}
        runOnUiThread { textView8.setText("Progress = $count") }
    }
}).start()
```

<table>
<tr><th>AutoCompleteTextView</th><th>MultiAutoCompleteTextView</th></tr>
<tr><td>AutocompleteTextView only offers suggestions about the whole sentence</td><td>MultiAutoCompleteTextView offers suggestions for every token in the sentence. You can specify what is the delimiter between tokens.</td></tr>
<tr><td>AutoCompleteTextView is used for selecting single Item</td><td>MultiAutoCompleteTextView is used for selecting multiple Items by using a delimiter(such as comma) in between them.</td></tr>
<tr><td>the “From:” field in example of email app, you would need to enforce only a single selection by the user from their configured email accounts.</td><td>If you were writing an email app, and you wanted the “To:” field to be an autocomplete field, getting matches from an address book, chances you want to allow the user to pick multiple recipients for a message, and would make this field a MultiAutoCompleteTextView</td></tr>
<tr><td><AutoCompleteTextView
android:id="@+id/autoCompleteTextView"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:completionThreshold="1"
android:hint="Enter City" /></td><td><MultiAutoCompleteTextView
android:id="@+id/multiAutoCompleteTextView"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:completionThreshold="1"
android:hint="Enter Skills" /></td></tr>
<tr><td>Example

var atv = findViewById<AutoCompleteTextView>(R.id.autoCompleteTextView)

var city = arrayOf("Ahmedabad","Rajkot","Rajsitapur","Randhanpur", "Ratanpar")

var adapter = ArrayAdapter<String>(this, android.R.layout.simple_dropdown_item_1line,city)
atv.setAdapter(adapter)</td><td>Example

var mtv = findViewById<MultiAutoCompleteTextView>(R.id.multiAutoCompleteTextView)

var skills = arrayOf("Java","Java Script","Kotlin","Android","Python", "PHP")

var adapter2 = ArrayAdapter<String>(this, android.R.layout.simple_list_item_checked, skills)
mtv.setAdapter(adapter2)

mtv.setTokenizer(MultiAutoCompleteTextView.CommaTokenizer())</td></tr>
<tr><td>Raj

Rajkot

Rajsitapur</td><td>PHP, Ja

Java

Java Script</td></tr>
</table>