<http://www.tutorialspoint.com/android/index.htm>

1. **Application Components**
   1. **Four Main Components**
      1. **Activities**

They dictate the UI and handle the user interaction to the smart phone 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.

* + 1. **Services**

They handle background processing associated with an application. A service is a component that runs in the background to perform long-running operations.

* + 1. **Broadcast Receivers**

They handle communication between Android OS and applications. Broadcast Receivers simply respond to broadcast messages from other applications or from the system. Each message is broadcaster as an Intent object

* + 1. **Content Providers**

They handle data and database management issues. 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.

* 1. **Other Components**
     1. **Fragments**

Represents a portion of user interface in an Activity.

* + 1. **Views**

UI elements that are drawn on-screen including buttons, lists forms etc.

* + 1. **Layouts**

View hierarchies that control screen format and appearance of the views.

* + 1. **Intents**

Messages wiring components together.

* + 1. **Resources**

External elements, such as strings, constants and drawable pictures.

* + 1. **Manifest**

Configuration file for the application. 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.

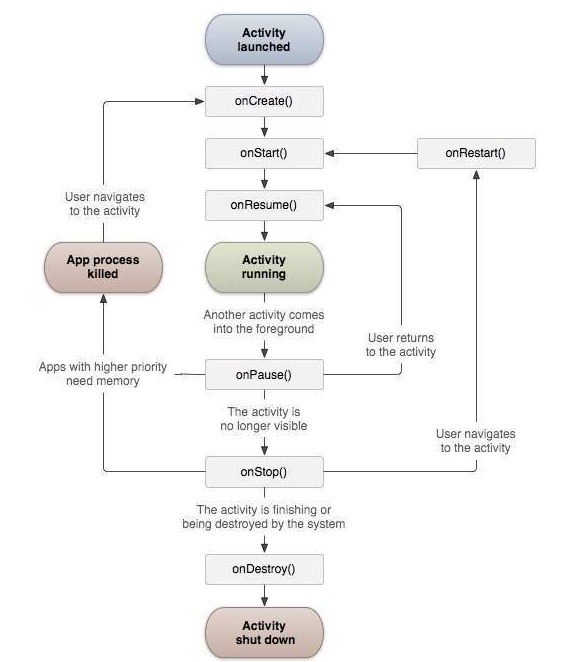
1. **Resource**
   1. **Directory**
      1. **anim/**
      2. **drawable/**
      3. **layout/**
      4. **menu/**
      5. **raw/**
      6. **values/**
      7. **xml/**
      8. **color/**
   2. **Alternative Resources**

The form of name is <resources\_name>-<config\_qualifier>/, Like drawable-hdpi/.

Your application should provide alternative resources to support specific device configurations.

1. **Activitys**

Activity like a window in windows programming.

****

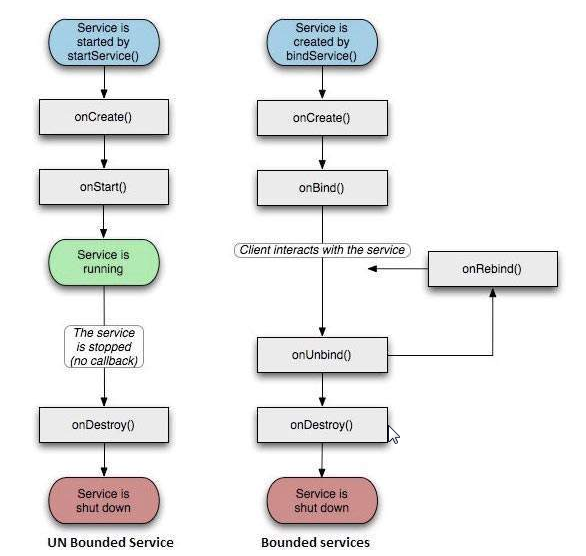
1. **Services**

A service is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed. A service can essentially take two states – Started and Bound.

Started: A service is started when an application component, such as an activity, starts it by calling startService(). Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.

Bound: A service is bound when an application component binds to it by calling bindService(). A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).

A service has life cycle callback methods that you can implement to monitor changes in the service's state and you can perform work at the appropriate stage.



1. **Broadcast Receivers**

Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents. 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 overriding the onReceive() method where each message is received as a Intent object parameter.

An application listens for specific broadcast intents by registering a broadcast receiver in AndroidManifest.xml file.

If you want your application itself should generate and send custom intents then you will have to create and send those intents by using the sendBroadcast() method inside your activity class.

1. **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. A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network.

To query a content provider, you specify the query string in the form of a URI which has following format:

<prefix>://<authority>/<data\_type>/<id>

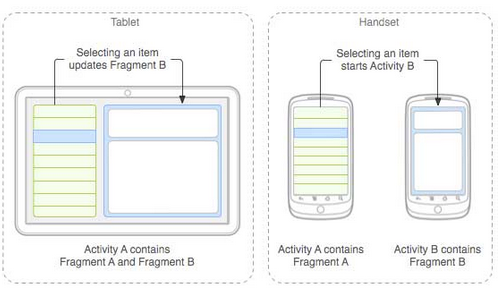
1. **Fragment**

A Fragment is a piece of an activity which enable more modular activity design. It will not be wrong if we say, a fragment is a kind of sub-activity.

* 1. Following are important points about fragment:
     1. A fragment has its own layout and its own behavior with its own life cycle callbacks.
     2. You can add or remove fragments in an activity while the activity is running.
     3. You can combine multiple fragments in a single activity to build a multi-plane UI.
     4. A fragment can be used in multiple activities.
     5. Fragment life cycle is closely related to the life cycle of its host activity which means when the activity is paused, all the fragments available in the activity will also be stopped.
     6. A fragment can implement a behavior that has no user interface component.
     7. Fragments were added to the Android API in Honeycomb version of Android which API version 11.
  2. **Applying Scenario**

Prior to fragment introduction, we had a limitation because we can show only a single activity on the screen at one given point in time. So we were not able to divide device screen and control different parts separately. But with the introduction of fragment we got more flexibility and removed the limitation of having a single activity on the screen at a time. Now we can have a single activity but each activity can comprise of multiple fragments which will have their own layout, events and complete life cycle.

Following is a typical example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.



The application can embed two fragments in Activity A, when running on a tablet-sized device. However, on a handset-sized screen, there's not enough room for both fragments, so Activity A includes only the fragment for the list of articles, and when the user selects an article, it starts Activity B, which includes the second fragment to read the article.

* 1. **Fragment Life Cycle**



* 1. **Types of Fragments**

Basically fragments are divided as three stages as shown below.

* + 1. Single frame fragments − Single frame fragments are using for hand hold devices like mobiles, here we can show only one fragment as a view.
    2. List fragments − fragments having special list view is called as list fragment
    3. Fragments transaction − Using with fragment transaction. we can move one fragment to another fragment
  1. **Fragment Transition**

When a scene changes, a Transition has two main responsibilities −

* + 1. Capture the state of each view in both the start and end scenes.
    2. Create an Animator based on the differences that will animate the views from one scene to the other.

1. **Intents and filter**

An Android Intent is an abstract description of an operation to be performed. It can be used with startActivity to launch an Activity, broadcastIntent to send it to any interested BroadcastReceiver components, and startService(Intent) or bindService(Intent, ServiceConnection, int) to communicate with a background Service.

An Intent object is a bundle of information which is used by the component that receives the intent as well as information used by the Android system.An Intent object is a bundle of information which is used by the component that receives the intent as well as information used by the Android system.

* 1. An Intent object can contain the following components based on what it is communicating or going to perform –
     1. Action

This is mandatory part of the Intent object and is a string naming the action to be performed — or, in the case of broadcast intents, the action that took place and is being reported. The action largely determines how the rest of the intent object is structured .

* + 1. Data

Adds a data specification to an intent filter. The specification can be just a data type (the mimeType attribute), just a URI, or both a data type and a URI.

* + 1. Category

The category is an optional part of Intent object and it's a string containing additional information about the kind of component that should handle the intent.

* + 1. Extras

This will be in key-value pairs for additional information that should be delivered to the component handling the intent. The extras can be set and read using the putExtras() and getExtras() methods respectively.

* + 1. Flags

These flags are optional part of Intent object and instruct the Android system how to launch an activity, and how to treat it after it's launched etc.

* + 1. Component Name
  1. **Types of Intents**
     1. **Explicit Intents**

Explicit intent going to be connected internal world of application, suppose if you wants to connect one activity to another activity, we can do this quote by explicit intent, below image is connecting first activity to second activity by clicking button.

* + 1. **Implicit Intents**

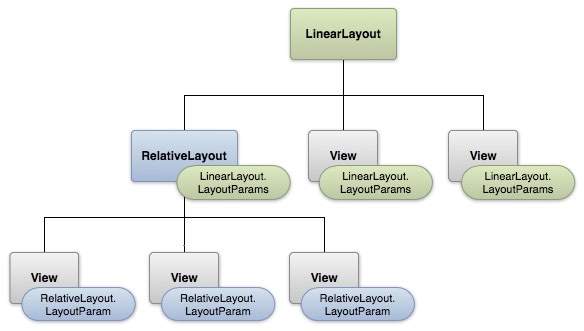
These intents do not name a target and the field for the component name is left blank. Implicit intents are often used to activate components in other applications.

1. **UI Layouts**

The basic building block for user interface is a View object which is created from the View class and occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components like buttons, text fields, etc.

The ViewGroup is a subclass of View and provides invisible container that hold other Views or other ViewGroups and define their layout properties.

At third level we have different layouts which are subclasses of ViewGroup class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using View/ViewGroup objects or you can declare your layout using simple XML file main\_layout.xml which is located in the res/layout folder of your project.



#### LAYOUT PARAMS

* 1. **Android Layout Types**
     1. **Linear Layout**

LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.

* + 1. **Relative Layout**

RelativeLayout is a view group that displays child views in relative positions.

* + 1. **Table Layout**

TableLayout is a view that groups views into rows and columns.

* + 1. **Absolute Layout**

AbsoluteLayout enables you to specify the exact location of its children.

* + 1. **Frame Layout**

Frame Layout is designed to block out an area on the screen to display a single item. Generally, FrameLayout should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other.

* + 1. **List View**

Android ListView is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database.

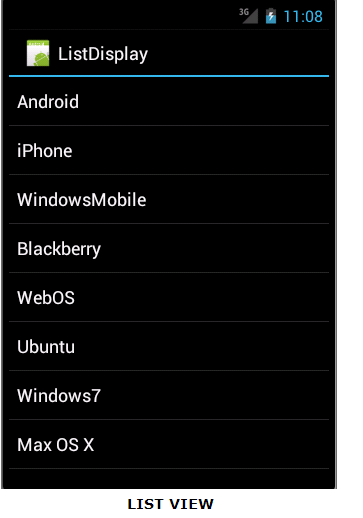
* + - 1. ArrayAdapter

You can use this adapter when your data source is an array.

* + - 1. SimpleCursorAdapter

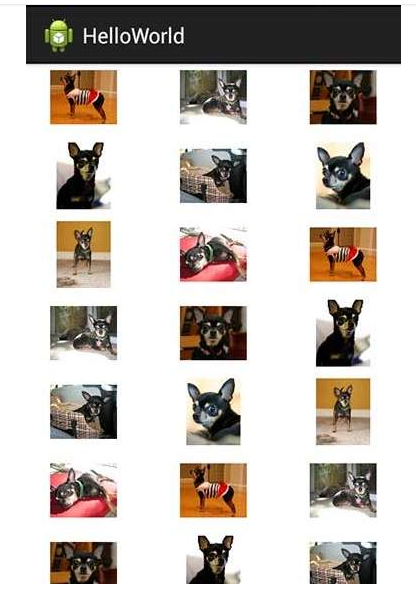
You can use this adapter when your data source is a database Cursor.

An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter holds the data and send the data to adapter view, the view can takes the data from adapter view and shows the data on different views like as spinner, list view, grid view etc.



* + 1. **Grid View**

GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.



Layout attributes:

Width and height unit: Here width and height are the dimension of the layout/view which can be specified in terms of dp (Density-independent Pixels), sp ( Scale-independent Pixels), pt ( Points which is 1/72 of an inch), px( Pixels), mm ( Millimeters) and finally in (inches).

<https://en.wikipedia.org/wiki/Device_independent_pixel>

A Device independent pixel (also: density-independent pixel, dip, dp) is a physical unit of measurement based on a co-ordinate system held by a computer and represents an abstraction of a pixel for use by an application that an underlying system then converts to physical pixels.

A typical use is to allow mobile device software to scale the display of information and user interaction to different screen sizes.

As dp is a physical unit it has an absolute value which can be measured in traditional units, e.g. for Android devices 1 dp equals 1/160 of inch or 0.15875 mm.

You can specify width and height with exact measurements but more often, you will use one of these constants to set the width or height –

• android:layout\_width=wrap\_content tells your view to size itself to the dimensions required by its content.

• android:layout\_width=fill\_parent tells your view to become as big as its parent view.

Gravity attribute plays important role in positioning the view object and it can take one or more (separated by '|') of the following constant values.

top, bottom, left, right, etc.

1. **Android UI Controls**
   1. **TextView**
   2. **EditText**
   3. **AutoCompleteTextView**

The AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing.

* 1. **Button**
  2. **ImageButton**
  3. **CheckBox**
  4. **ToggleButton**

An on/off button with a light indicator.

* 1. **RadioButton**
  2. **RadioGroup**
  3. **ProgressBar**
  4. **Spinner**

A drop-down list that allows users to select one value from a set.

* 1. **TimePicker**

The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode.

* 1. **DatePicker**

The DatePicker view enables users to select a date of the day.

1. **Event Handling**
   1. **Event Listeners**

An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.

* 1. **Event Listeners Registration**

Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.

Though there are several tricky ways to register your event listener for any event, but I'm going to list down only top 3 ways, out of which you can use any of them based on the situation.

* + 1. **Using an Anonymous Inner Class**
    2. **Activity class implements the Listener interface.**
    3. **Using Layout file activity\_main.xml to specify event handler directly.**
  1. **Event Handlers**

When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

**Touch Mode**

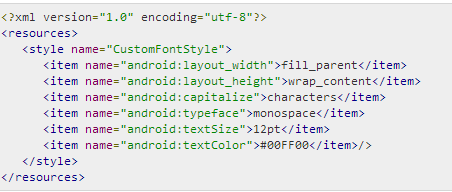
Users can interact with their devices by using hardware keys or buttons or touching the screen. Touching the screen puts the device into touch mode. The user can then interact with it by touching the on-screen virtual buttons, images, etc. You can check if the device is in touch mode by calling the View class’s isInTouchMode() method.

1. **Styles and Themes**

A style resource defines the format and look for a UI. A style can be applied to an individual View (from within a layout file) or to an entire Activity or application (from within the manifest file).

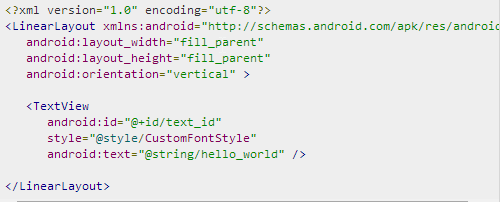
**Defining Styles**

A style is defined in an XML resource that is separate from the XML that specifies the layout. This XML file resides under res/values/ directory of your project and will have <resources> as the root node which is mandatory for the style file. The name of the XML file is arbitrary, but it must use the .xml extension.You can define multiple styles per file using <style> tag but each style will have its name that uniquely identifies the style. Android style attributes are set using <item> tag as shown below –



**Using Styles**

Once your style is defined, you can use it in your XML Layout file using style attribute as follows –



**Style Inheritance**

Android supports style Inheritance in very much similar way as cascading style sheet in web design. You can use this to inherit properties from an existing style and then define only the properties that you want to change or add

**Using a Custom Nine-Patch With Buttons**

A nine-patch drawable is a special kind of image which can be scaled in width and height while maintaining its visual integrity. Nine-patches are the most common way to specify the appearance of Android buttons, though any drawable type can be used.



1. Save this bitmap as /res/drawable/my\_nine\_patch.9.png
2. Define a new style
3. Apply the new button style to the buttonStyle attribute of your custom theme

**Themes**

A theme is nothing but an Android style applied to an entire Activity or application, rather than an individual View.

Thus, when a style is applied as a theme, every View in the Activity or application will apply each style property that it supports.

To set a theme for all the activities of your application, open the AndroidManifest.xml file and edit the <application> tag to include the android:theme attribute with the style name.

<application android:theme="@style/CustomFontStyle">

But if you want a theme applied to just one Activity in your application, then add the android:theme attribute to the <activity> tag only

<activity android:theme="@style/CustomFontStyle">

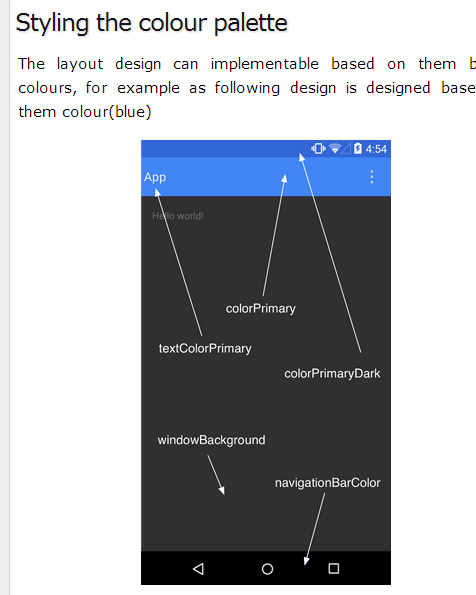
There are number of default themes defined by Android which you can use directly or inherit them using parent attribute as follows

<style name="CustomTheme" parent="android:Theme.Light">

...

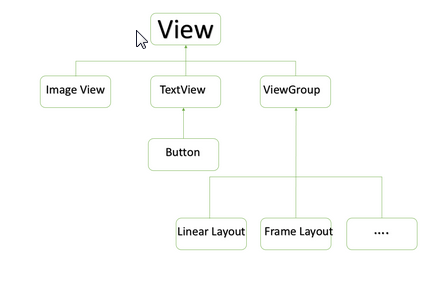
</style>

**Styling the colour palette**



1. **Custom Component (UI Control)**

If you only need to make small adjustments to an existing widget or layout, you can simply subclass the widget or layout and override its methods which will give you precise control over the appearance and function of a screen element.



**Custom Component with Custom Attributes**

When you want to create your own attribute from scratch.

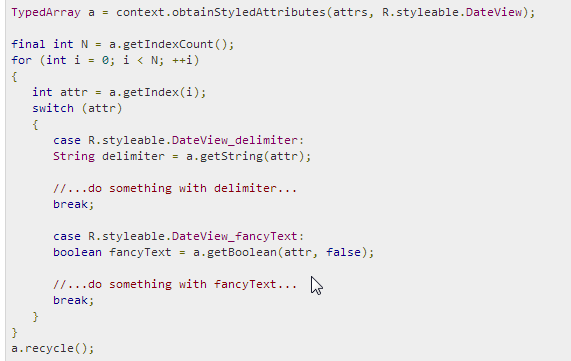
Step 1

The first step to enable us to use our custom attributes is to define them in a new xml file under res/values/ and call it attrs.xml. Let's have a look on an example attrs.xml file –



Step 2

Your second step will be to read these attributes from Layout XML file and set them for the component. This logic will go in the constructors that get passed an AttributeSet, since that is what contains the XML attributes. To read the values in the XML, you need to first create a TypedArray from the AttributeSet, then use that to read and set the values as shown in the below example code –



Step 3

Finally you can use your defined attributes in your Layout XML file as follows −



The important part is xmlns:custom="http://schemas.android.com/apk/res/com.example.compoundview".

1. Drag and Drop

View can receive drag event to do drag-drop movement - View.OnDragEventListener

Can build a shadow for the process of drag -View.DragShadowBuilder.

1. **Notification**

Create and Send Notifications

* 1. **Step 1 - Create Notification Builder**

As a first step is to create a notification builder using NotificationCompat.Builder.build().

* 1. **Step 2 - Setting Notification Properties**

Once you have Builder object, you can set its Notification properties using Builder object as per your requirement. But this is mandatory to set at least following –

A small icon, set by setSmallIcon()

A title, set by setContentTitle()

Detail text, set by setContentText()

* 1. **Step 3 - Attach Actions**

This is an optional part and required if you want to attach an action with the notification. An action allows users to go directly from the notification to an Activity in your application, where they can look at one or more events or do further work.

The action is defined by a PendingIntent containing an Intent that starts an Activity in your application. To associate the PendingIntent with a gesture, call the appropriate method of NotificationCompat.Builder. For example, if you want to start Activity when the user clicks the notification text in the notification drawer, you add the PendingIntent by calling setContentIntent().

A PendingIntent object helps you to perform an action on your applications behalf, often at a later time, without caring of whether or not your application is running.

* 1. **Step 4 - Issue the notification**

Finally, you pass the Notification object to the system by calling NotificationManager.notify() to send your notification. Make sure you call NotificationCompat.Builder.build() method on builder object before notifying it. This method combines all of the options that have been set and return a new Notification object.