Layouts

Layout

Layouts can be declared in two ways:

- **Declare UI elements in XML**
  Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
- **Instantiate layout elements at runtime**
  Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

Either or both of these methods can be used for declaring your application's UI.

Advantage of Declaring Application UI in XML

- it enables you to better separate the presentation of your application from the code that controls its behavior.
- Your UI descriptions are external to your application code, which means that you can modify or adapt it without having to modify your source code and recompile.
- Declaring the layout in XML makes it easier to visualize the structure of your UI, so it's easier to debug problems.

XML Layouts

- The XML vocabulary for declaring UI elements closely follows the structure and naming of the classes and methods, where element names correspond to class names and attribute names correspond to methods.
- Not all vocabulary is identical.

Writing XML Layout

- Each layout file must contain exactly one root element, which must be a View or ViewGroup object.
- Once you've defined the root element, you can add additional layout objects or widgets as child elements to gradually build a View hierarchy that defines your layout.
- After you've declared your layout in XML, save the file with the .xml extension, in your Android project's res/layout/ directory, so it will properly compile.

Writing XML Layout : Example

```
<?xml version="1.0" encoding="utf-8"?>
   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
          android:layout_width="fill_parent"
          android:layout_height="fill_parent"
          android:orientation="vertical" >
     <TextView android:id="@+id/text"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="Hello, I am a TextView" />
     <Button android:id="@+id/button"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Hello, I am a Button" />
   </LinearLayout>
```

Loading XML Resource

- When you compile your application, each XML layout file is compiled into a View resource.

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView.(R.layout.main_layout);
    }
```

Attributes

- Every View and ViewGroup object supports their own variety of XML attributes.
- Some attributes are specific to a View object.
- Some are common to all View objects, because they are inherited from the root View class (like the id attribute).
- And, other attributes are considered "layout parameters," which are attributes that describe certain layout orientations of the View object, as defined by that object's parent ViewGroup object.

Attributes : id

- Any View object may have an integer ID associated with it, to uniquely identify the View within the tree.
- When the application is compiled, this ID is referenced as an integer, but the ID is typically assigned in the layout XML file as a string, in the id attribute.
- This is an XML attribute common to all View objects.

```
android:id="@+id/my_button"
```

- The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource. The plus-symbol (+) means that this is a new resource name that must be created and added to our resources

Attributes : id

In order to create views and reference them from the application, a common pattern is to:

1. Define a view/widget in the layout file and assign it a unique ID:
2. Then create an instance of the view object and capture it from the layout (typically in the onCreate() method):

```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
-----------------------------------------------------------------------------------------
Button myButton = (Button) findViewById(R.id.my_button);
```

Layout Parameters

- XML layout attributes named layout_*something* define layout parameters for the View that are appropriate for the ViewGroup in which it resides.
- Every ViewGroup class implements a nested class that extends ViewGroup.LayoutParams.
  This subclass contains property types that define the size and position for each child view.
- Each child element must define LayoutParams that are appropriate for its parent, though it may also define different LayoutParams for its own children.

Layout Parameters

- All view groups include a width and height (layout_width and layout_height), and each view is required to define them. Many LayoutParams also include optional margins and borders.
- More often, you will use one of these constants to set the width or height:
  - *wrap_content* tells your view to size itself to the dimensions required by its content
  - *fill_parent* tells your view to become as big as its parent view group will allow.

Layout Position

- The geometry of a view is that of a rectangle.
- A view has a location, expressed as a pair of *left* and *top* coordinates, and two dimensions, expressed as a width and a height. The unit for location and dimensions is the pixel.
- It is possible to retrieve the location of a view by invoking the methods getLeft() and getTop().
- These methods both return the location of the view relative to its parent.
- Several methods are provided to reduce unnecessary computations. E.g. getRight() and getBottom().

Size, Padding and Margins

- The size of a view is expressed with a width and a height. A view actually possess two pairs of width and height values.
- The first pair is known as *measured width* and *measured height*. These dimensions define how big a view wants to be within its parent.
  - The measured dimensions can be obtained by calling getMeasuredWidth() and getMeasuredHeight().
- The second pair is simply known as *width* and *height*, or sometimes *drawing width* and *drawing height*.
  These dimensions define the actual size of the view on screen, at drawing time and after layout.
  - The width and height can be obtained by calling getWidth() and getHeight().

Size, Padding and Margins

- Padding can be set using the setPadding(int, int, int, int) method and queried by calling getPaddingLeft(), getPaddingTop(), getPaddingRight() and getPaddingBottom().
- Even though a view can define a padding, it does not provide any support for margins. However, view groups provide such a support.

Layout Examples

- Linear Layout
- Relative Layout
- Table Layout
- Tab Layout
- List View
- Grid View

Relative Layout

- RelativeLayout is a ViewGroup that displays child View elements in relative positions.
- The position of a View can be specified as relative to sibling elements (such as to the left-of or below a given element) or in positions relative to the RelativeLayout area.

Example

```
<?xml version="1.0" encoding="utf-8"?>
    <RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
       android:layout_width="fill_parent"
       android:layout_height="fill_parent">
       <TextView
          android:id="@+id/label"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:text="Type here:"/>
       <EditText
          android:id="@+id/entry"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:background="@android:drawable/editbox_background"
          android:layout_below="@id/label"/>
```

Example

```
<Button
    android:id="@+id/ok"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/entry"
    android:layout_alignParentRight="true"
    android:layout_marginLeft="10dip"
    android:text="OK" />
 <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toLeftOf="@id/ok"
    android:layout_alignTop="@id/ok"
    android:text="Cancel" /
</RelativeLayout>
```

Table Layout

- TableLayout is a ViewGroup that displays child View elements in rows and columns.

```
<?xml version="1.0" encoding="utf-8"?>
    <TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
       android:layout_width="fill_parent"
       android:layout_height="fill_parent"
       android:stretchColumns="1">

       <TableRow>
          <TextView
             android:layout_column="1"
             android:text="Open..."
             android:padding="3dip" />
          <TextView
             android:text="Ctrl-O"
             android:gravity="right"
             android:padding="3dip" />
       </TableRow>
    </TableLayout>
```

Grid View

- GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.
- The grid items are automatically inserted to the layout using a ListAdapter.

Refer the example SimpleGridViewDemo

Tab Layout

- To create a tabbed UI, you need to use a TabHost and a TabWidget.
- The TabHost must be the root node for the layout, which contains both the TabWidget for displaying the tabs and a FrameLayout for displaying the tab content.

Refer the example TabLayoutDemo

ListView

- ListView is a ViewGroup that creates a list of scrollable items. The list items are automatically inserted to the list using a ListAdapter.

Refer the demo ***ListViewDemo***