Introduction to Mobile Computing (day3)

Dieudonne U

Android - UI Layouts

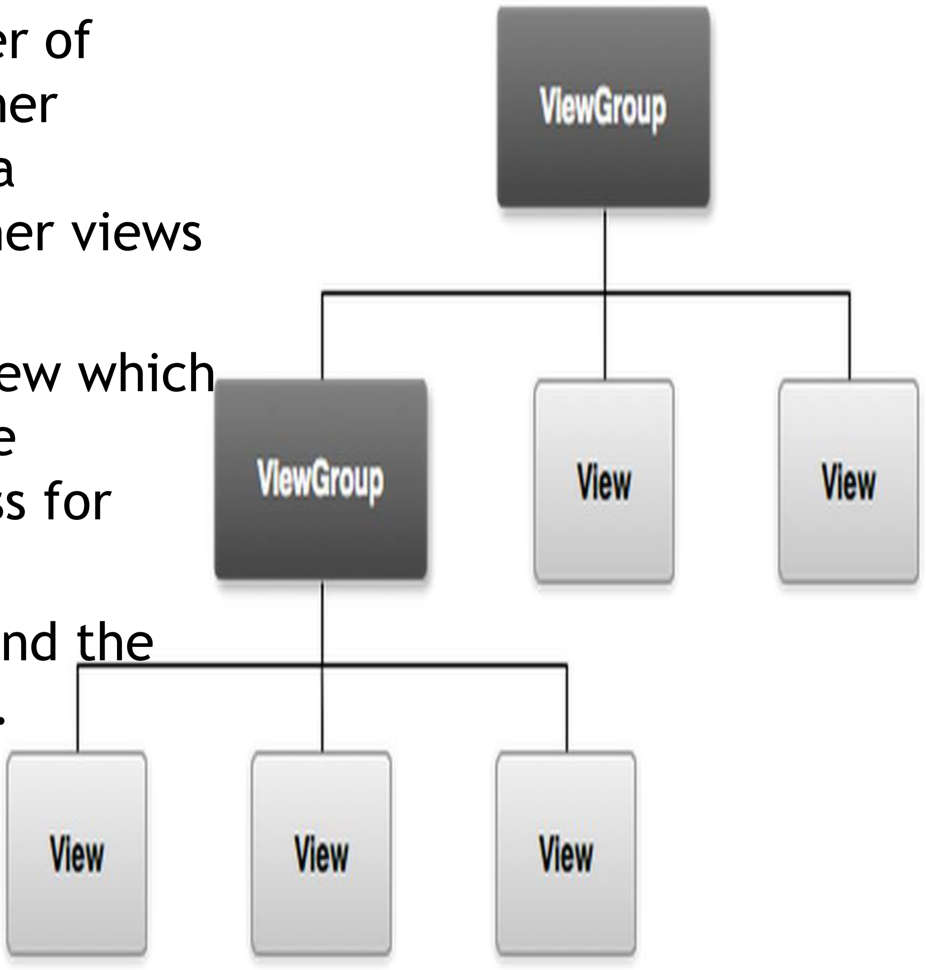
- Creating your GUI based on layouts defined in XML file.
- A layout may contain any type of widgets such as buttons, labels, textboxes, and so on.
- Try to understand what is an Activity, Intent.
- Build a simple mobile app from scratch
    - Start a new activity
    - Build GUI
    - Add EventListeners to UI elements
    - Learn about the lifecycle of the app
    - Compile and run the app.

ViewGroup

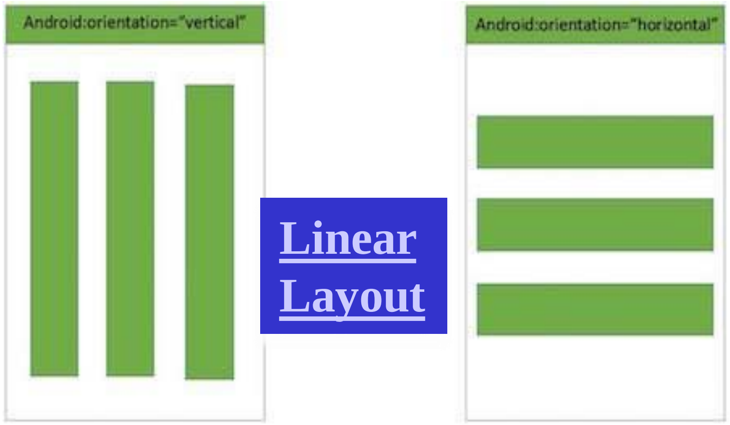
- A ViewGroup is a special view that can contain other views (called children.) The view group is the base class for layouts and views containers. This class also defines the ViewGroup.LayoutParams class which serves as the base class for layouts parameters.

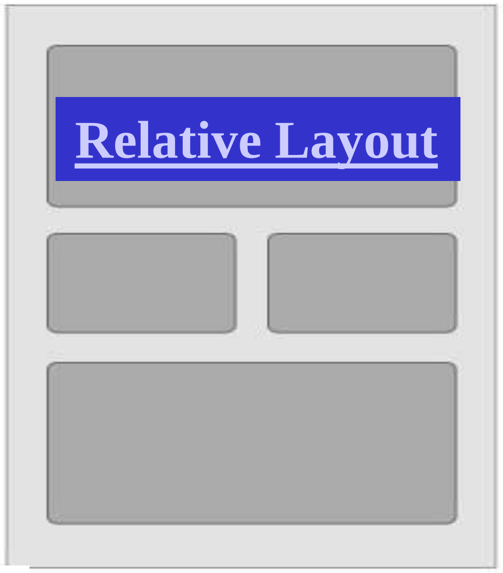
  - ViewGroup classes include-
    - LinearLayout, TableLayout, RelativeLayout and others.
  - Every of these ViewGroup classes has LayoutParams inner class.
  - The base class for these LayoutParams is ViewGroup.LayoutParams.

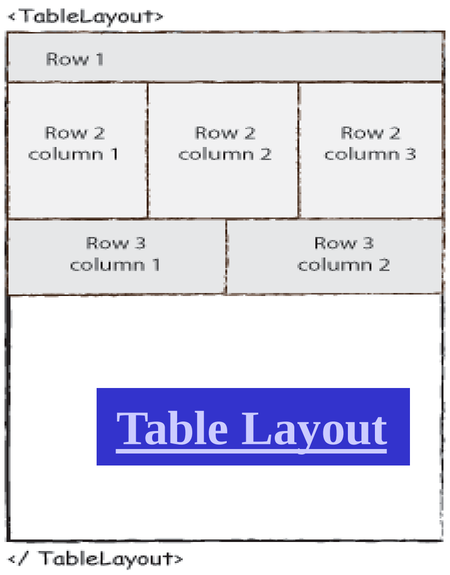
- View is a basic building block of UI (User Interface) in android. A view is a small rectangular box which responds to user inputs. Eg: EditText, Button, CheckBox, etc..
- ViewGroup is a invisible container of other views (child views) and other viewgroups. Eg: LinearLayout is a viewgroup which can contain other views in it.
- ViewGroup is a special kind of view which is extended from View as its base class. ViewGroup is the base class for layouts.
- as name states View is singular and the group of Views is the ViewGroup.

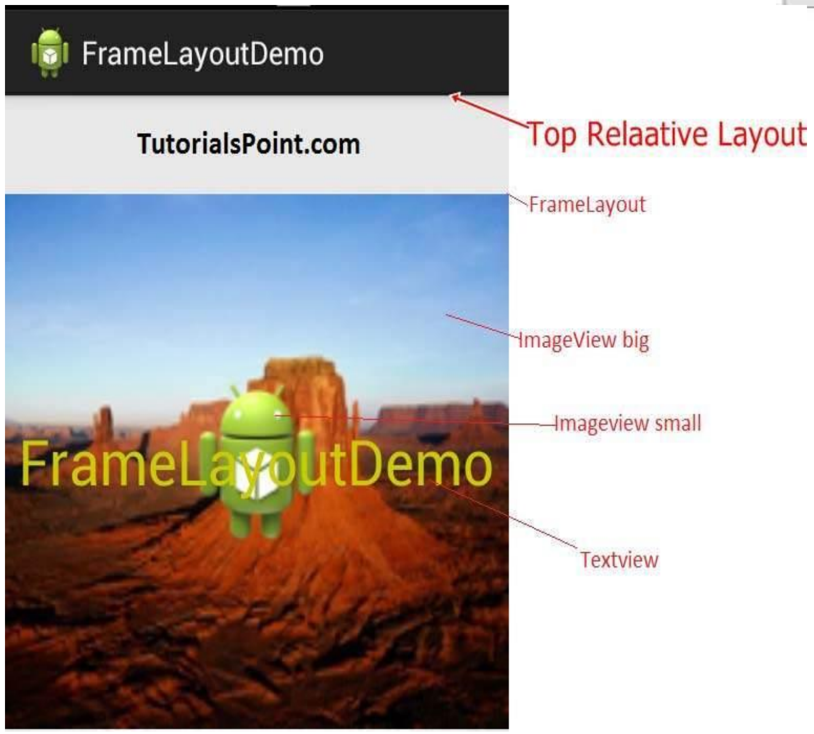


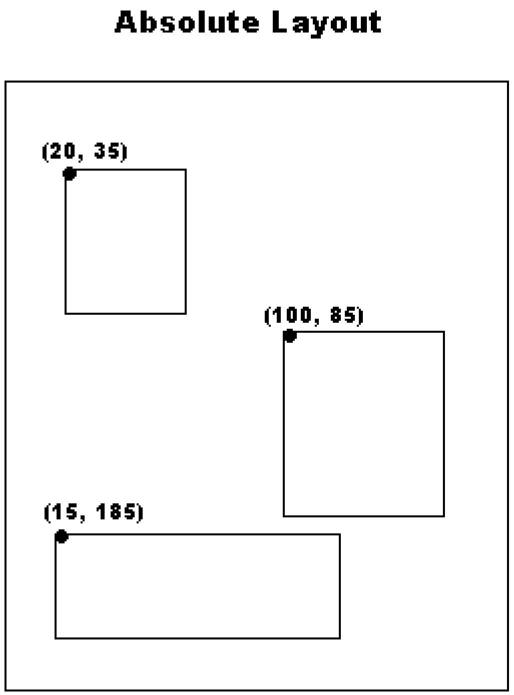
Layout & Description
Linear Layout   LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
Relative Layout   RelativeLayout is a view group that displays child views in relative positions.
Table Layout   TableLayout is a view that groups views into rows and columns.
Frame Layout   The FrameLayout is a placeholder on screen that you can use to display a single view.
List View   ListView is a view group that displays a list of scrollable items.
ConstraintLayout   GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

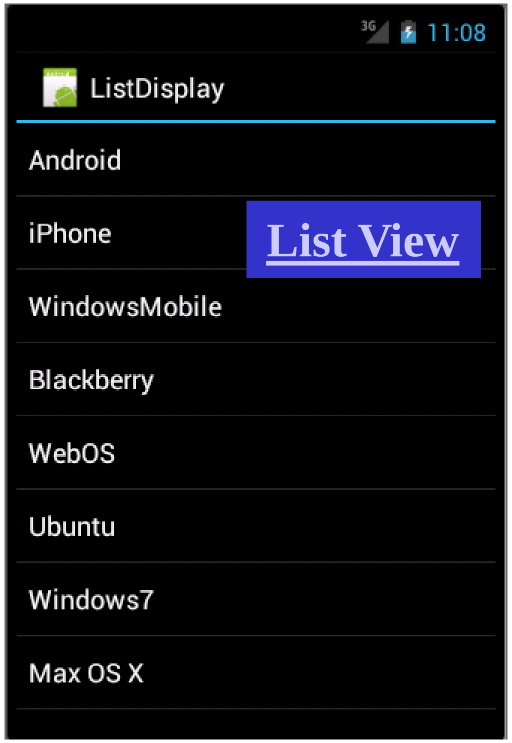

Android:orientation="vertical"
Linear Layout
Android:orientation="horizontal"











Android

iPhone

WindowsMobile

Blackberry

WebOS

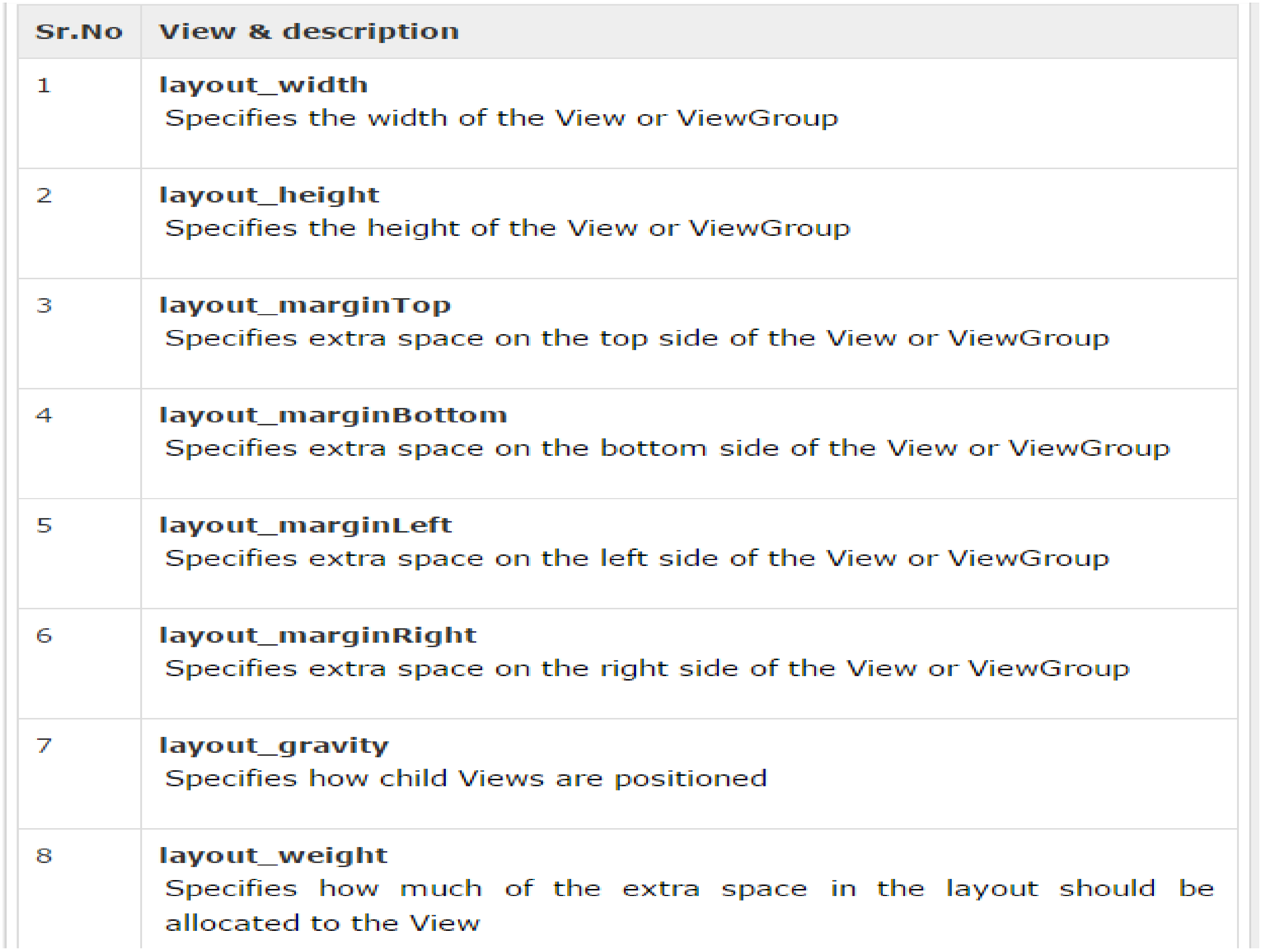
Ubuntu

Windows7

Max OS X

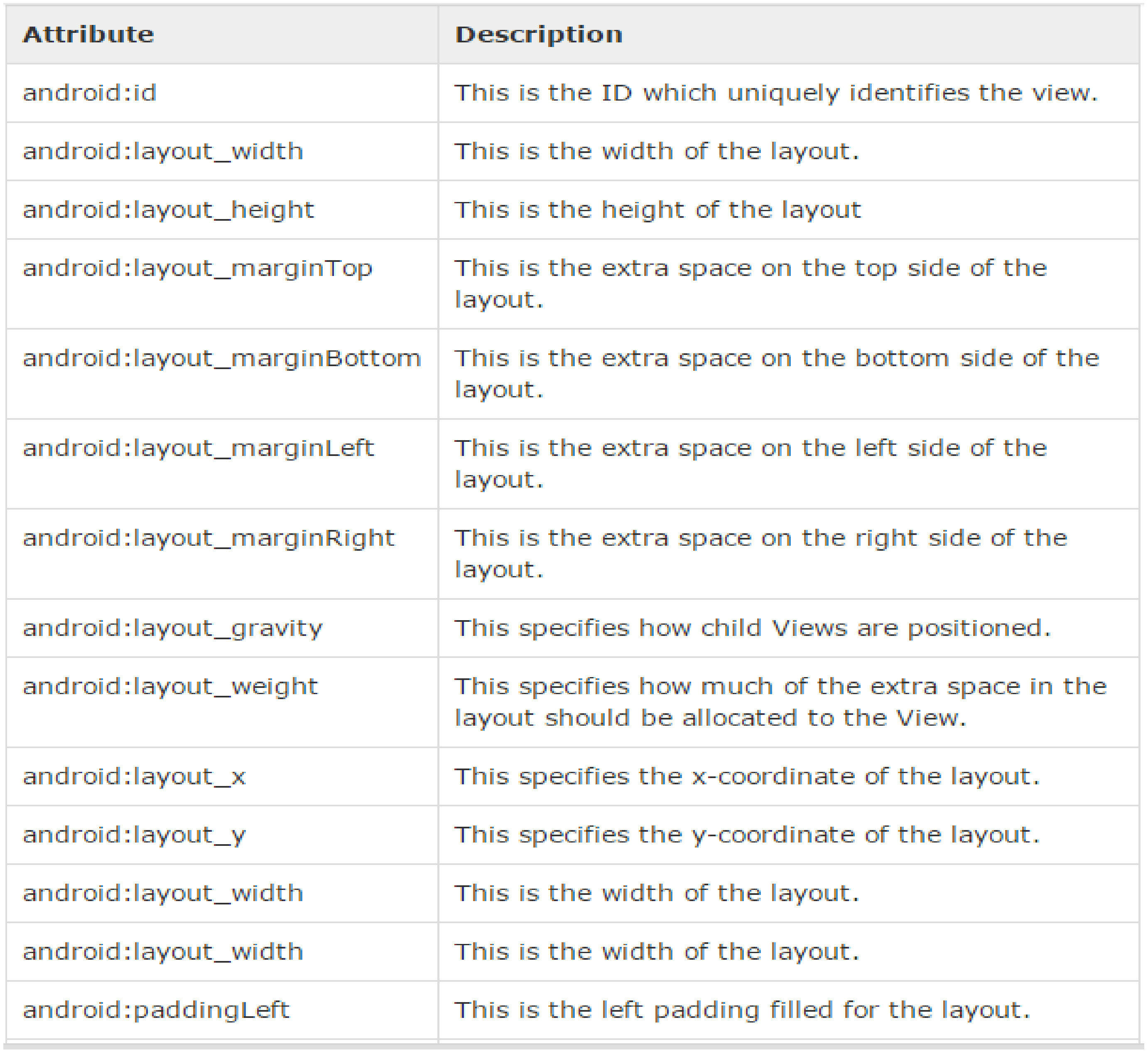
Layout & Description
ConstraintLayout The ConstraintLayout is a powerful new class, similar to RelativeLayout. It allows us to lay out child views using ‘constraints’ to define position based relationships between different views found in our layout. The aim of the ConstraintLayout is to help reduce the number of nested views, which will improve the performance of our layout files. The layout class also makes it easier for us to define layouts than when using a RelativeLayout as we can now anchor any side of a view with any side of another, rather than having to place a whole view to any side of another.

Views and ViewGroups attributes



Sr.No	View & description
1	**layout_width** Specifies the width of the View or ViewGroup
2	**layout_height** Specifies the height of the View or ViewGroup
3	**layout_marginTop** Specifies extra space on the top side of the View or ViewGroup
4	**layout_marginBottom** Specifies extra space on the bottom side of the View or ViewGroup
5	**layout_marginLeft** Specifies extra space on the left side of the View or ViewGroup
6	**layout_marginRight** Specifies extra space on the right side of the View or ViewGroup
7	**layout_gravity** Specifies how child Views are positioned
8	**layout_weight** Specifies how much of the extra space in the layout should be allocated to the View

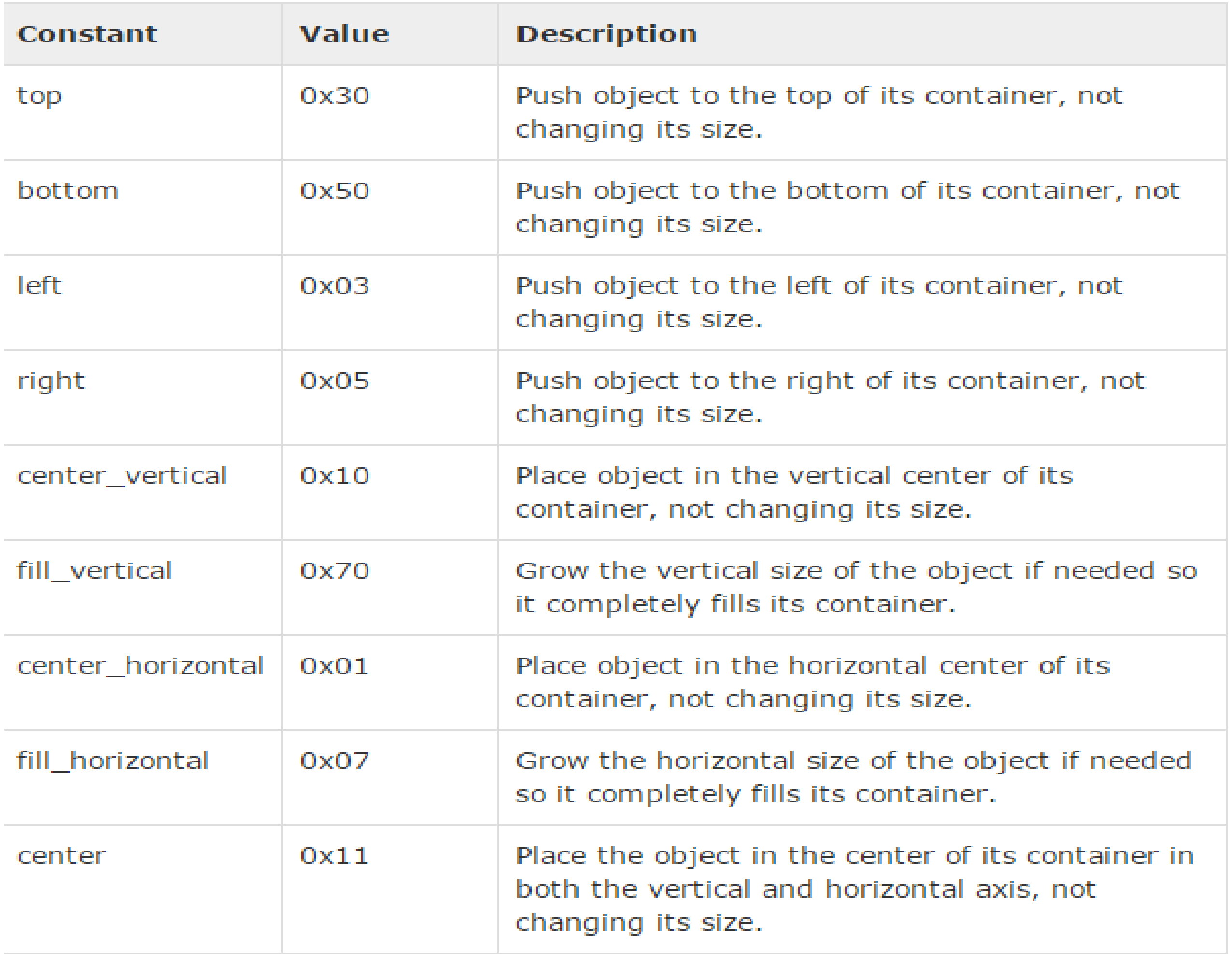
Layout Attributes



Attribute	Description
android:id	This is the ID which uniquely identifies the view.
android:layout_width	This is the width of the layout.
android:layout_height	This is the height of the layout
android:layout_marginTop	This is the extra space on the top side of the layout.
android:layout_marginBottom	This is the extra space on the bottom side of the layout.
android:layout_marginLeft	This is the extra space on the left side of the layout.
android:layout_marginRight	This is the extra space on the right side of the layout.
android:layout_gravity	This specifies how child Views are positioned.
android:layout_weight	This specifies how much of the extra space in the layout should be allocated to the View.
android:layout_x	This specifies the x-coordinate of the layout.
android:layout_y	This specifies the y-coordinate of the layout.
android:layout_width	This is the width of the layout.
android:layout_width	This is the width of the layout.
android:paddingLeft	This is the left padding filled for the layout.

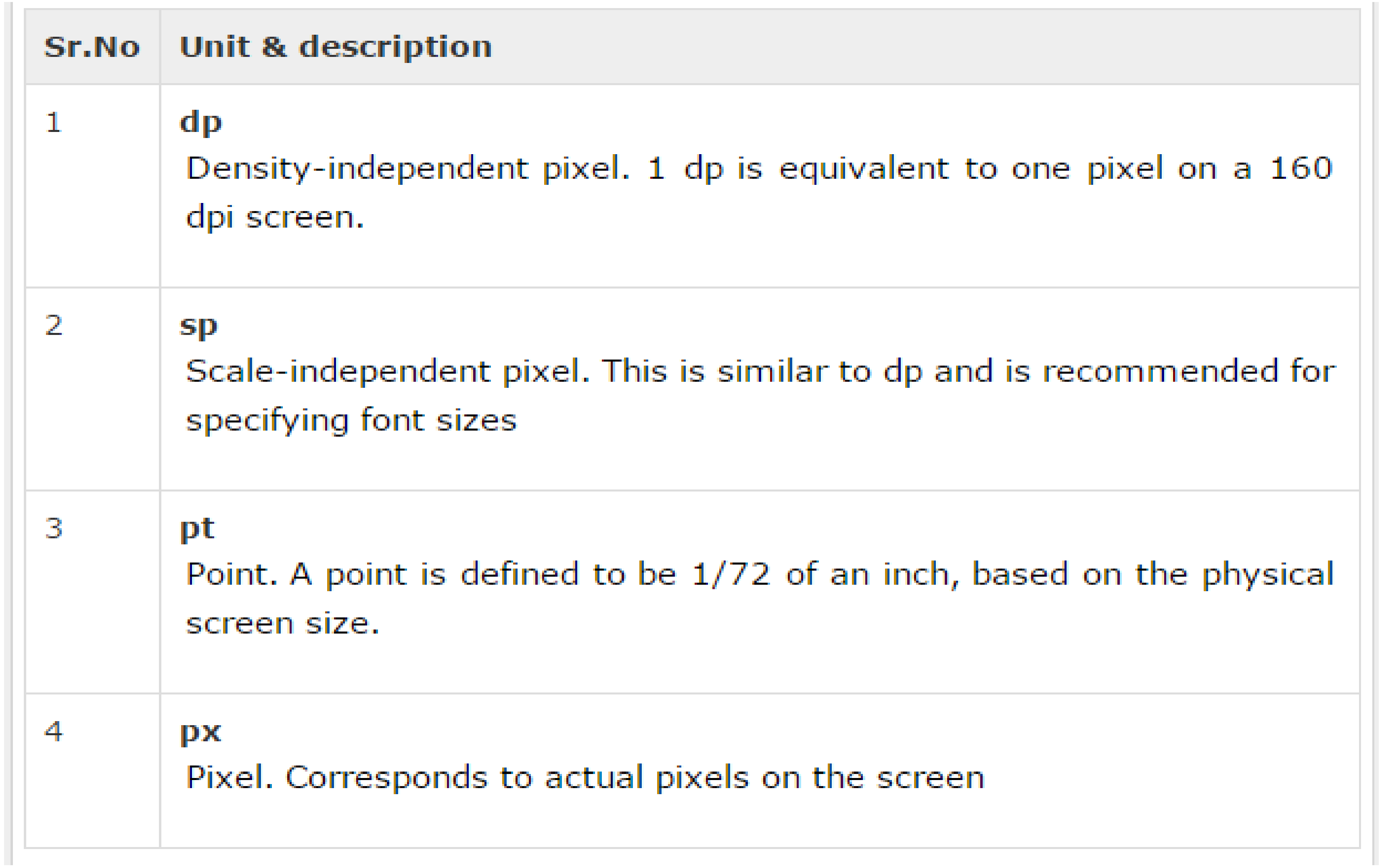
Width and height with exact measurements

- **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.



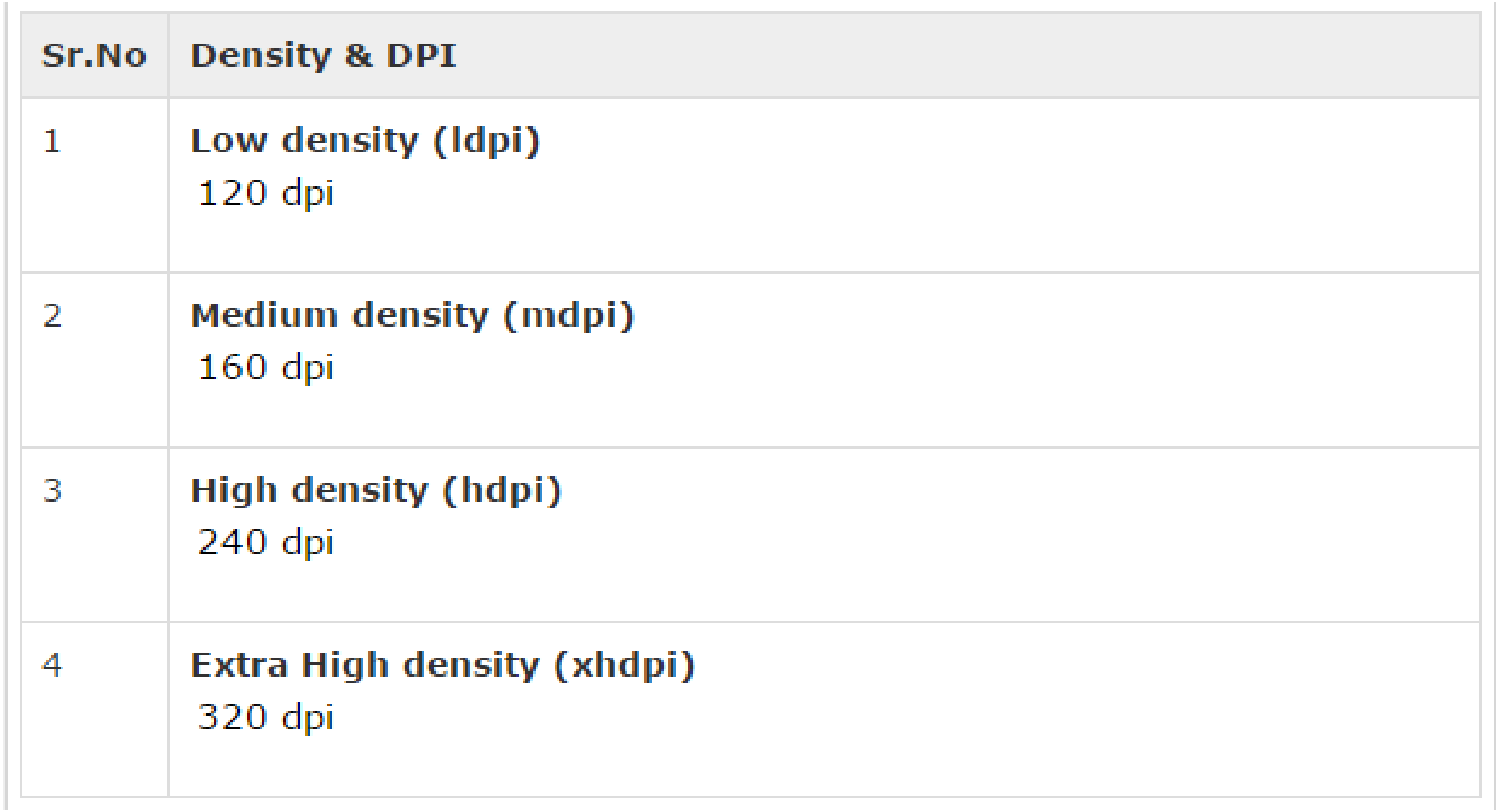
Constant	Value	Description
top	0x30	Push object to the top of its container, not changing its size.
bottom	0x50	Push object to the bottom of its container, not changing its size.
left	0x03	Push object to the left of its container, not changing its size.
right	0x05	Push object to the right of its container, not changing its size.
center_vertical	0x10	Place object in the vertical center of its container, not changing its size.
fill_vertical	0x70	Grow the vertical size of the object if needed so it completely fills its container.
center_horizontal	0x01	Place object in the horizontal center of its container, not changing its size.
fill_horizontal	0x07	Grow the horizontal size of the object if needed so it completely fills its container.
center	0x11	Place the object in the center of its container in both the vertical and horizontal axis, not changing its size.

Units of Measurement



Sr.No	Unit & description
1	**dp** Density-independent pixel. 1 dp is equivalent to one pixel on a 160 dpi screen.
2	**sp** Scale-independent pixel. This is similar to dp and is recommended for specifying font sizes
3	**pt** Point. A point is defined to be 1/72 of an inch, based on the physical screen size.
4	**px** Pixel. Corresponds to actual pixels on the screen

Screen Densities



Sr.No	Density & DPI
1	**Low density (ldpi)** 120 dpi
2	**Medium density (mdpi)** 160 dpi
3	**High density (hdpi)** 240 dpi
4	**Extra High density (xhdpi)** 320 dpi

Optimizing layouts guidelines

- Avoid unnecessary nesting
- Avoid using too many Views
- Avoid deep nesting

Event Handling

- Events are a useful way to collect data about a user's interaction with interactive components of Applications.
  - Like button presses or screen touch etc.
- There are following three concepts related to Android Event Management −
- Event Listeners
- Event Listeners Registration
- Event Handlers

Event Handling

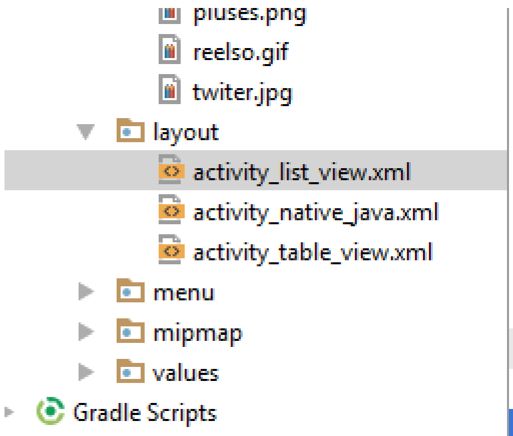
- **Event Listeners** − An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.
- **Event Listeners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

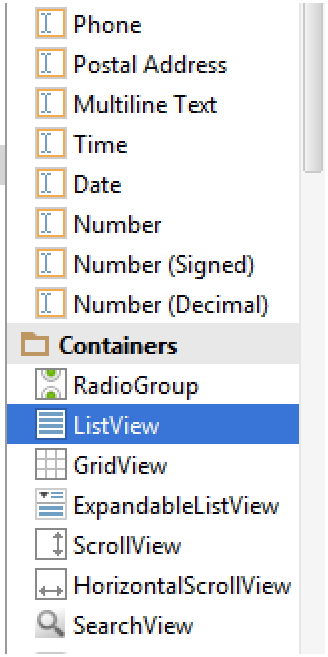
Some Event Listeners & Event Handlers

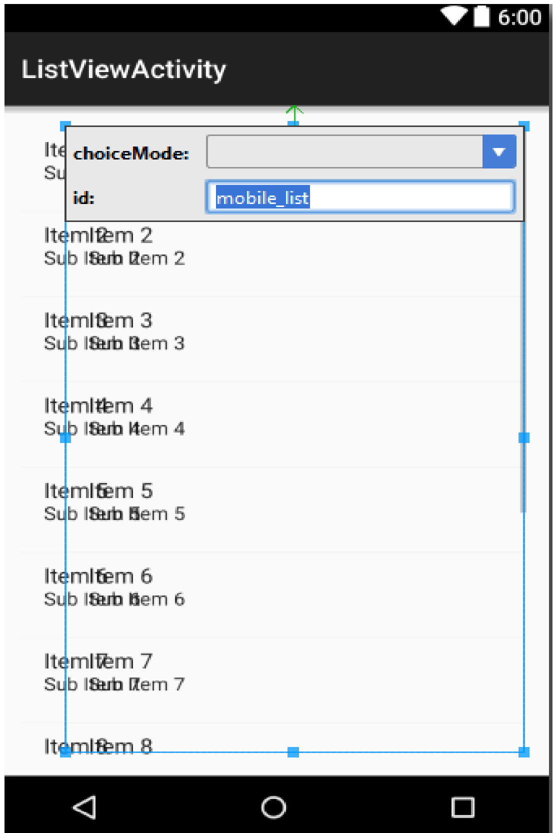
Event Handler	Event Listener & Description
onClick()	**OnClickListener()**This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. You will use onClick() event handler to handle such event.
onLongClick()	**OnLongClickListener()**This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event.
onFocusChange()	**OnFocusChangeListener()**This is called when the widget looses its focus ie. user goes away from the view item. You will use onFocusChange() event handler to handle such event.
onKey()	**OnFocusChangeListener()**This is called when the user is focused on the item and presses or releases a hardware key on the device. You will use onKey() event handler to handle such event.
onTouch()	**OnTouchListener()**This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use onTouch() event handler to handle such event.

Event Handling Examples

- Create a new activity named as **ListViewActivity**
- Modify *src/ListViewActivity.java* file to add click event listeners and handlers for the two buttons defined.
- Add the following code (copy paste the code from FirstAssignment -- *ListViewActivity*)
- Open the design view of activity_list_view and drop a listView from the pallet
- Double click the list and set Id to: mobile_list





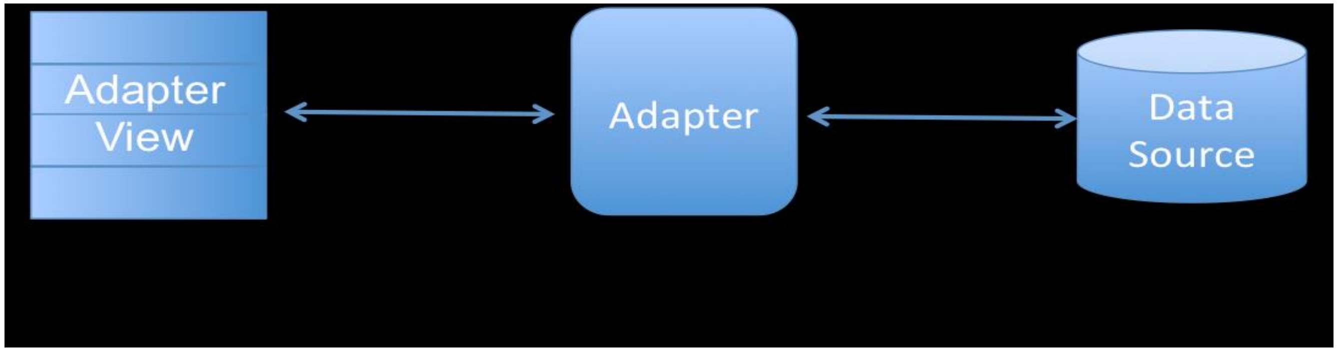


choiceMode:

id: mobile_list

Adapter

- Android Adapter is a bridge between the View (e.g. ListView) and the underlying data for that view.
- An adapter manages the data and adapts the data to the individual rows (listItems) of the view.
- We bind the adapter with Android listview via *setAdapter* method.
- To interact with the view, adapters call the getView() method which returns a view for each item within the adapter view. This is a listitem. The layout format and the corresponding data for an item within the adapter view are set in the getView() method.
- Once we have a reference to the view, we can get the data from the data source either in an Array list or a cursor. We can then bind the data to the view items. All this is done in getView Method.



Basic ListView with ArrayAdapter

ArrayAdapter

- Whenever you have a list of single items which is backed by an array, you can use ArrayAdapter. For instance, list of countries or names.
- By default, ArrayAdapter expects a Layout with single TextView.
- **Android ListView Example (ArrayAdapter)**

- ListActivity has a default layout that consists of a single, full-screen list in the center of the screen.
- Our layout MUST contain a ListView object with the id "@android:id/list". (NOT android:id="@+id/list").
- if we define a ListView with the android:id attribute set to @+id/list then it will throw below RuntimeException

Refer to the link below for other examples:

1.http://www.vogella.com/tutorials/AndroidListView/article.html

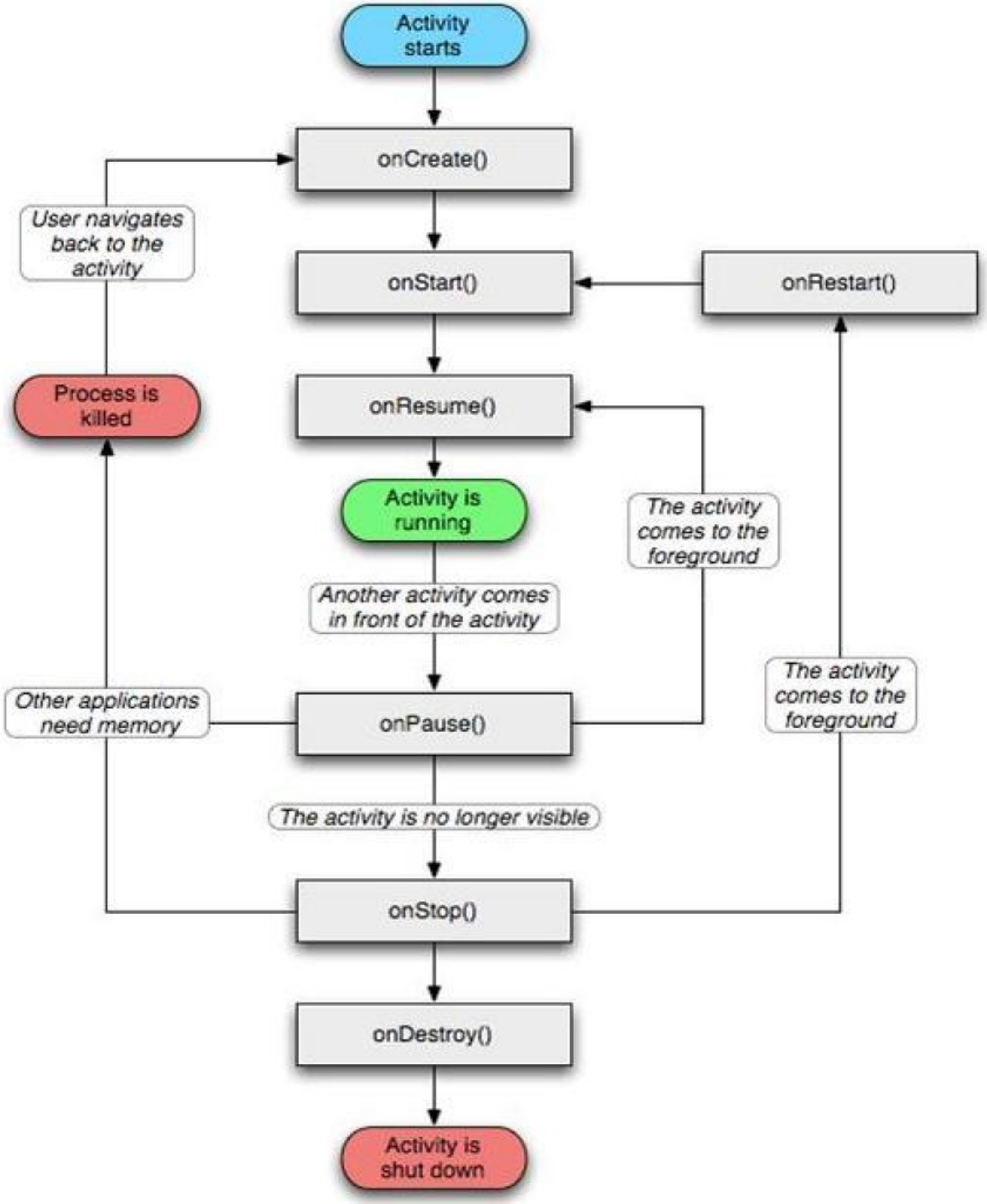
2. http://www.mkyong.com/android/android-listview-example/

Event Handling Examples

- Then go back to TableViewActivity and change the onCreate method by adding (if you copied from code just uncomment the code onclick event handling) the bellow code
- Notice the clickBtn (this is the Id of the button named Click! we added)

```
Button button = (Button)findViewById(R.id.ClickBtn);
    button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent i = new Intent(v.getContext(), ListViewActivity.class);
           startActivityForResult(i,0);
        }
    });
```

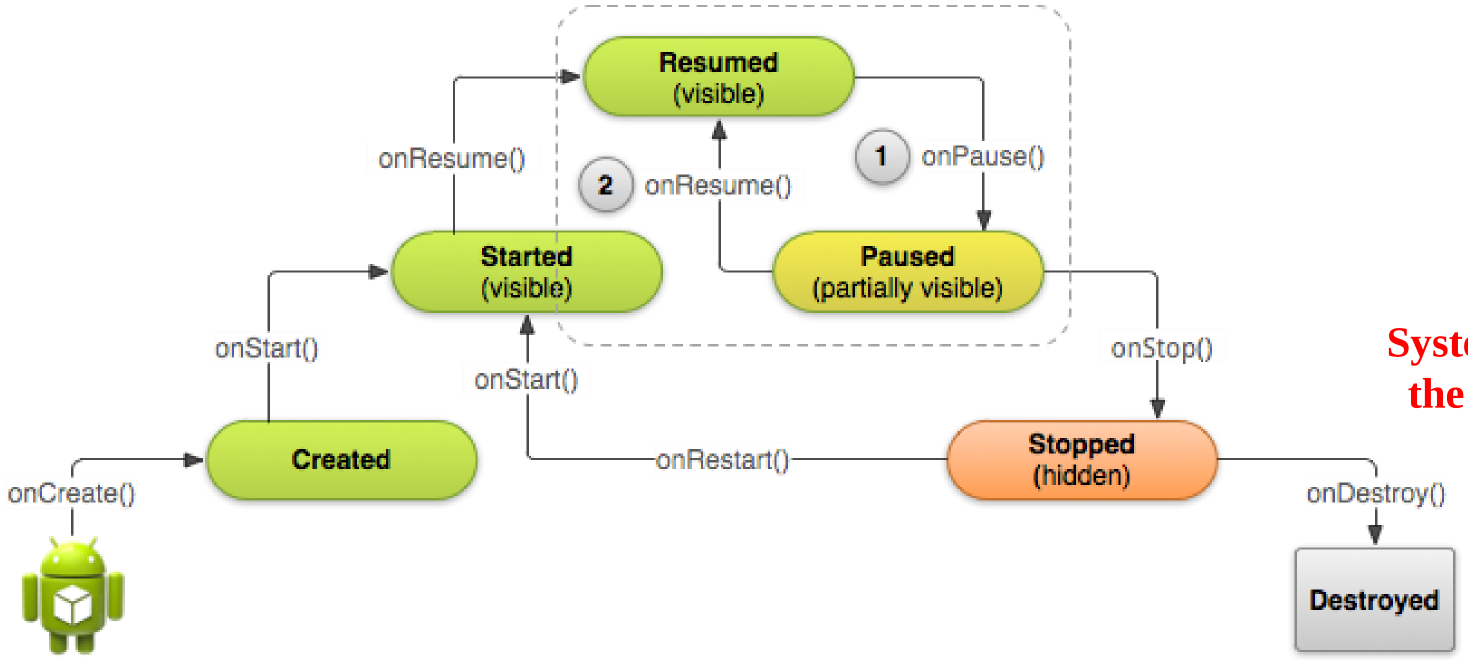
Application lifecycle


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Stopping and Starting an Activity

- How is stopping an activity defined?
  - If the activity is not visible
  - Press back or home button
  - Start a new activity or receive a phone call

What should you do before an activity stops
save state of the application?
release resources



System destroys the activity