Android Development

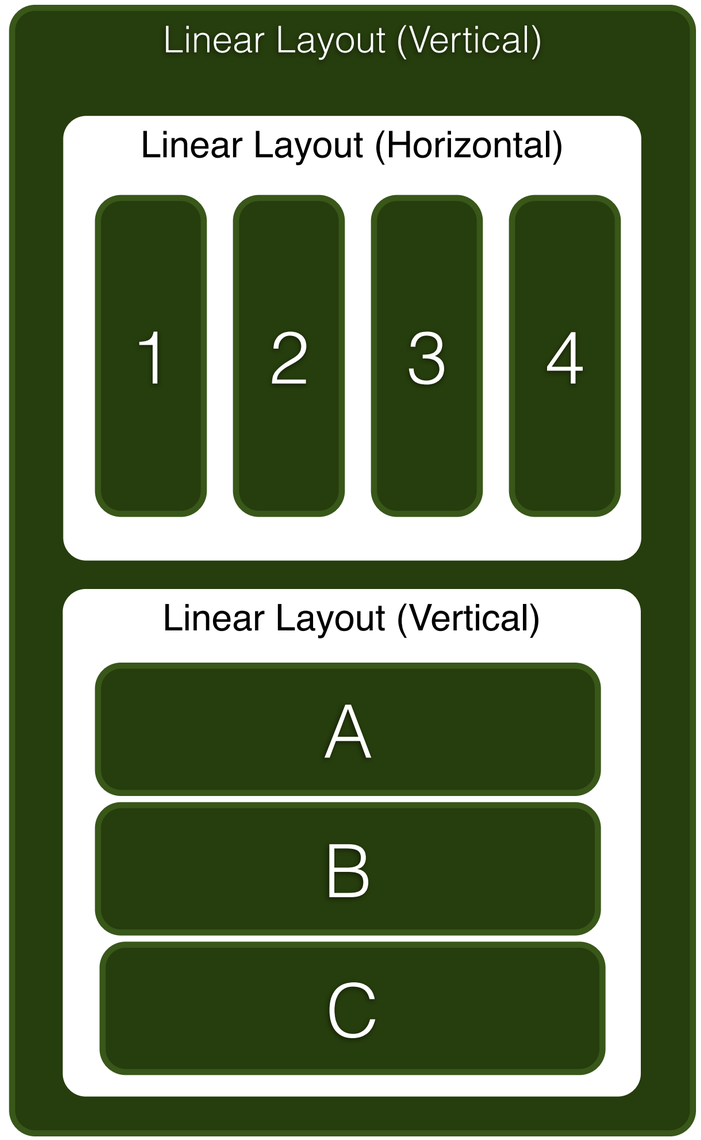
Lecture 3
Android Graphical User Interface 1

Lecture Summary

- Linear Layout
- Relative Layout
- Table Layout
- Grid View
- Tab Layout
- List View
- Custom List View Element
- Fragments

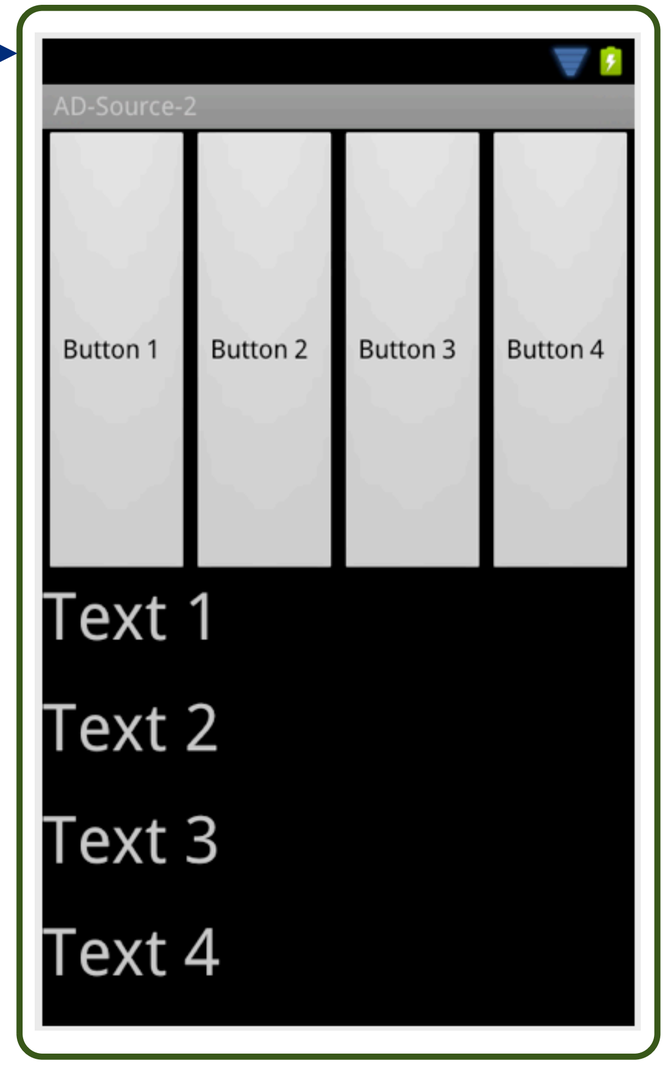
Linear Layout

- A Layout that arranges its children in a single column or a single row.
- Available space is divided among layout children
- The direction of the row can be set by calling setOrientation() or through XML using “`android:orientation="vertical"`”.
- You can also specify gravity, which specifies the alignment of all the child elements by calling setGravity() or specify that specific children grow to fill up any remaining space in the layout by setting the weight member ofLinearLayout.LayoutParams.
- The default orientation is horizontal.



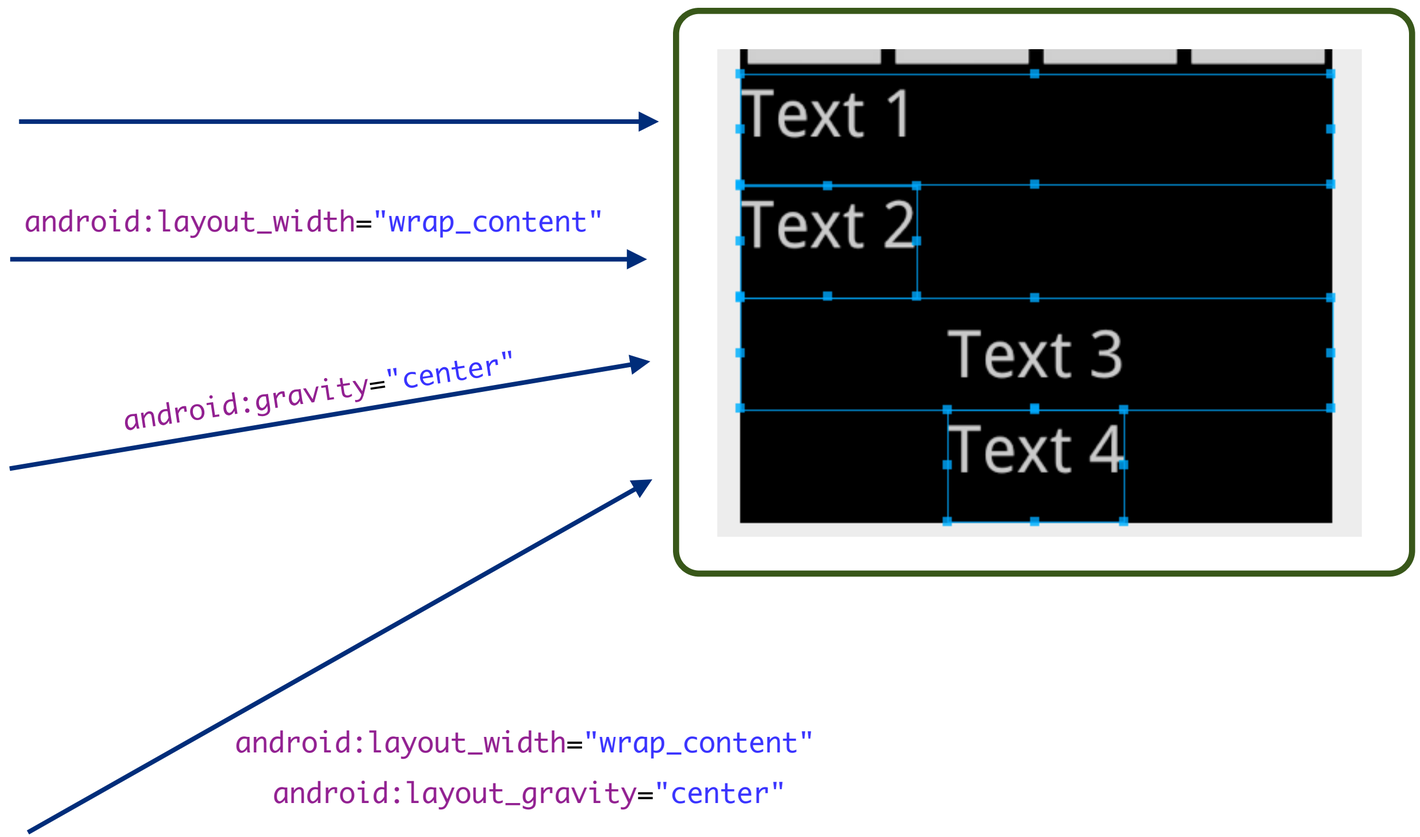
Linear Layout

```
 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1">
    <Button
        android:text="Button 1"
        android:gravity="center_horizontal"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="1"
    />
    <Button ... />
    <Button ... />
    <Button ... />
</LinearLayout>
<LinearLayout
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:layout_weight="1">
  <TextView
      android:text="Text 1"
      android:textSize="15pt"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_weight="1"/>
  <TextView ... />
  <TextView ... />
  <TextView ... />
  <TextView ... />

  </LinearLayout>
</LinearLayout>
```



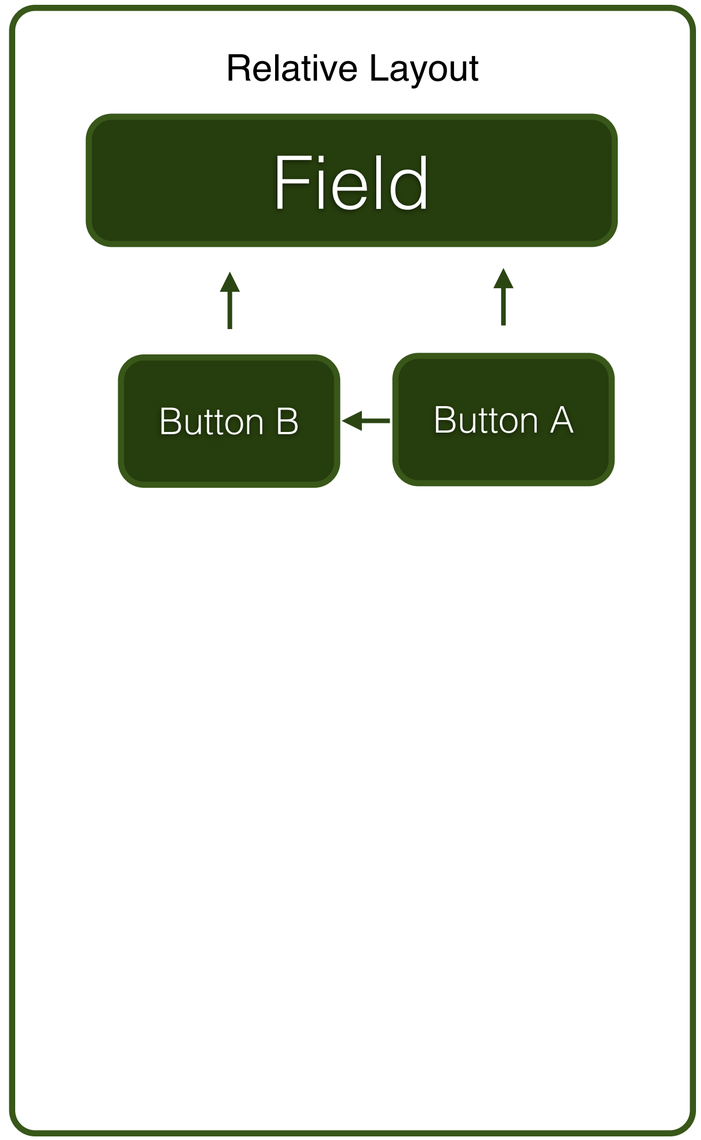
Layout Parameters

```
<LinearLayout
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:layout_weight="1">
  <TextView
       android:text="Text 1"
       android:textSize="15pt"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:layout_weight="1"/>
  <TextView
       android:text="Text 2"
       android:textSize="15pt"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_weight="1"/>
  <TextView
       android:text="Text 3"
       android:textSize="15pt"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:gravity="center"
       android:layout_weight="1"/>
  <TextView
       android:text="Text 4"
       android:textSize="15pt"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_gravity="center"
       android:layout_weight="1"/>
</LinearLayout>
```

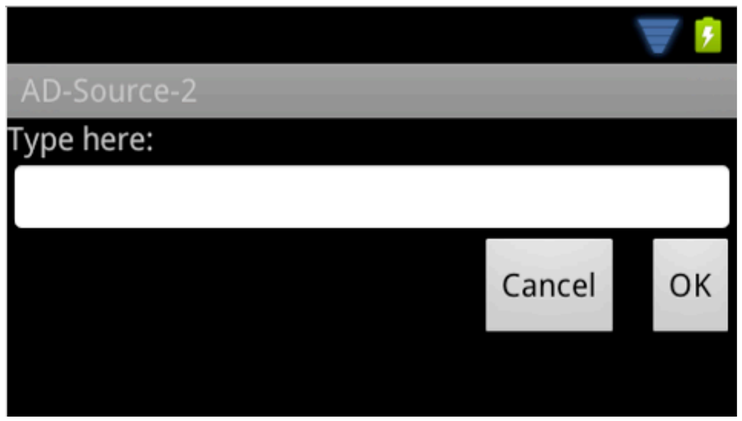


Relative Layout

- 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 (such as aligned to the bottom, left of center).
- A RelativeLayout is a very powerful utility for designing a user interface because it can eliminate nested ViewGroups. If you find yourself using several nested LinearLayout groups, you may be able to replace them with a single RelativeLayout.
- When using a RelativeLayout, you can use **android:layout_*** attributes, such as **layout_below**, **layout_alignParentRight**, and **layout_toLeftOf** to describe how you want to position each View. Each one of these attributes define a different kind of relative position. Some attributes use the resource ID of a sibling View to define its own relative position.



Relative Layout



android:layout_below="@id/label"

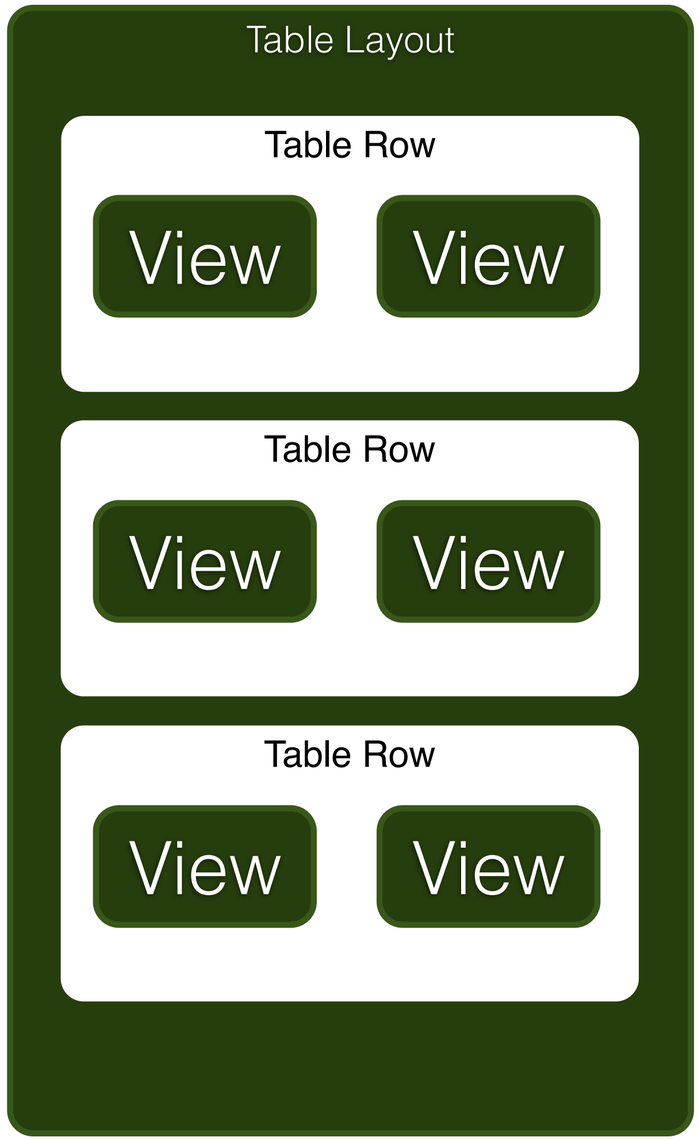
android:layout_below="@id/entry"
android:layout_alignParentRight="true"

android:layout_toLeftOf="@id/ok"
android:layout_alignTop="@id/ok"

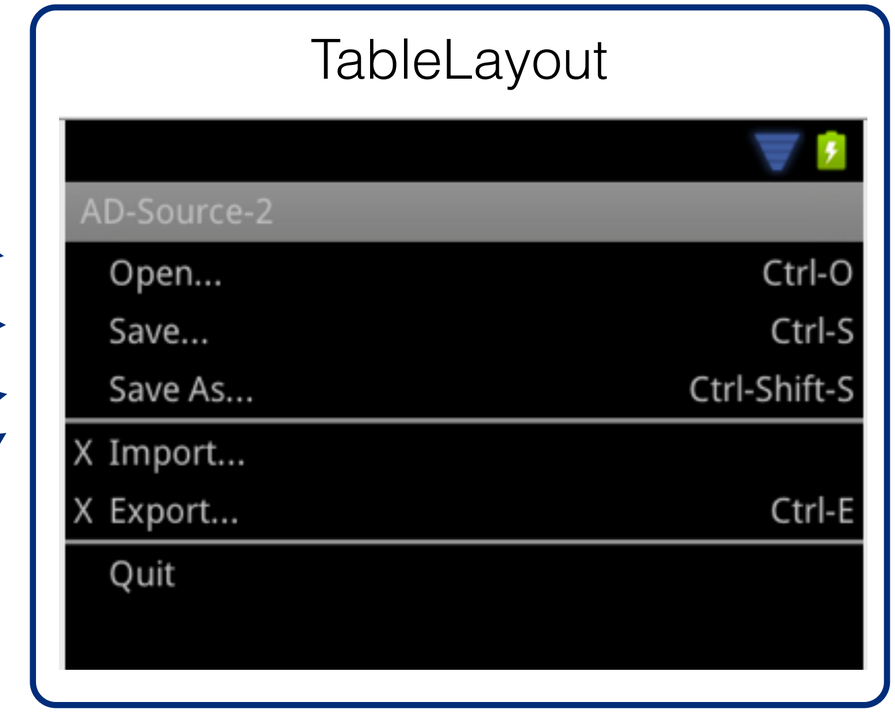
```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        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"/>
    <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.
- Has a structure similar to an HTML table. The TableLayout element is like the HTML **<table>** element; TableRow is like a **<tr>** element.
- In each cell you can use any kind of View element arranged like columns with horizontal linear layout.

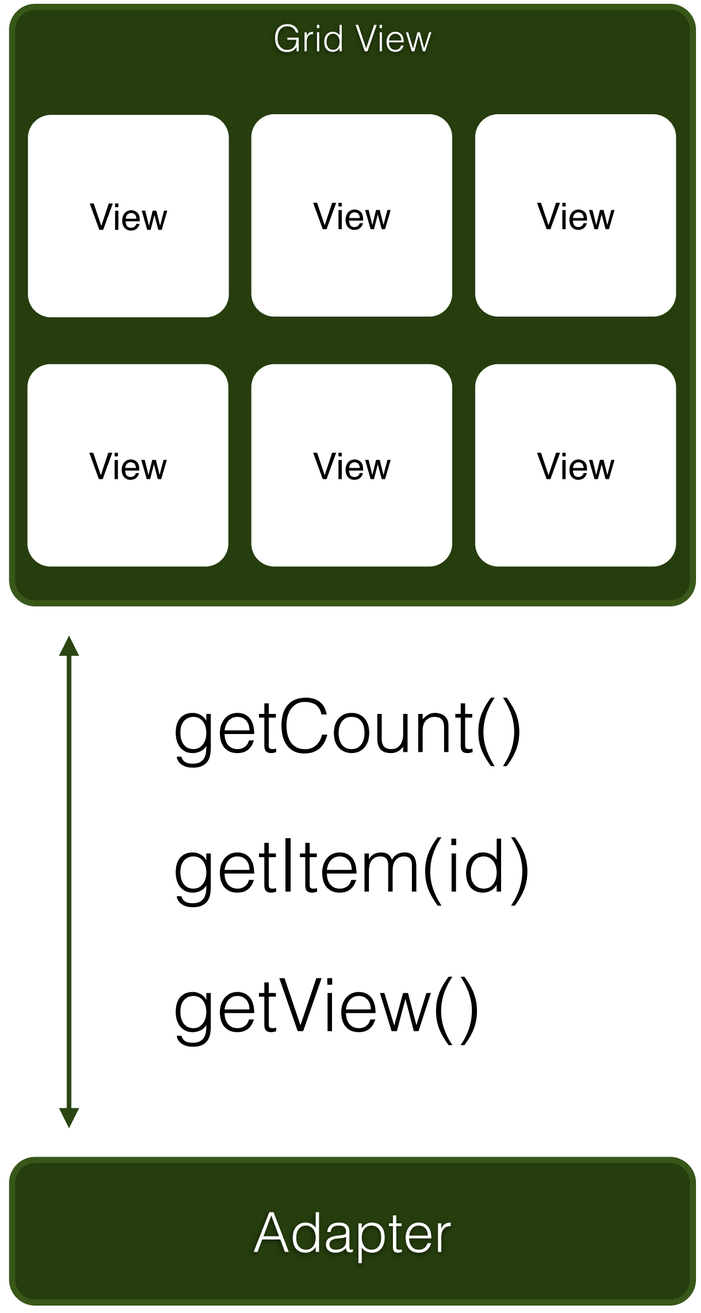


```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
android:layout_height="fill_parent"
    android:stretchColumns="1">
    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="Open..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-O"
            android:gravity="right"/>
    </TableRow>
    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="Save..." />
        <TextView
            android:text="Ctrl-S"
            android:gravity="right"/>
    </TableRow>
    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="Save As..."/>
        <TextView
            android:text="Ctrl-Shift-S"
            android:gravity="right"/>
    </TableRow>
    <View
        android:layout_height="2dip"
        android:background="#FF909090" />
.....

</TableLayout>
```



Grid View

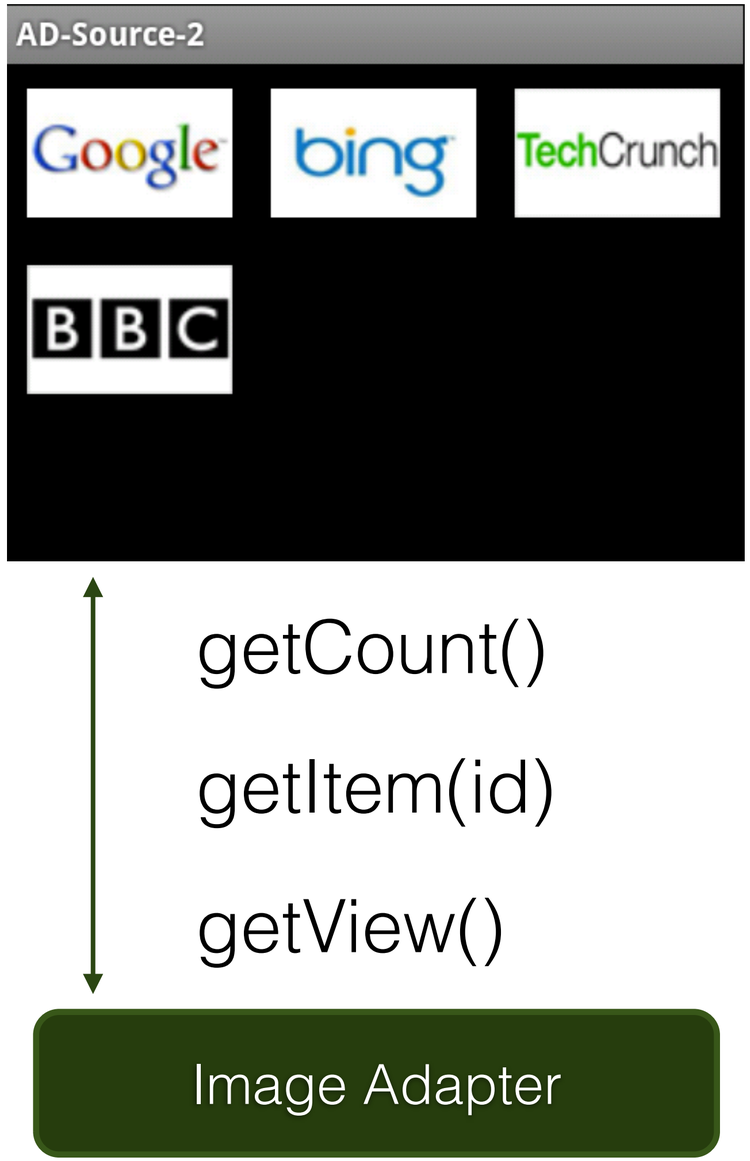
- GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid. The grid items are automatically inserted to the layout using a Adapter.
- An Adapter (BaseAdapter or ListAdapter) is the bridge between a List/Grid View and the data that fill the view.
- Grid View implementation is a perfect example of Model View Controller approach.
  - Grid View shows elements.
  - The Adapter manages the model defining information structure and providing data to the view.
  - The method setOnItemClickListener(new OnItemClickListener() {..}); allow to define the controller behavior when a click event is added to the event queue.



Grid View

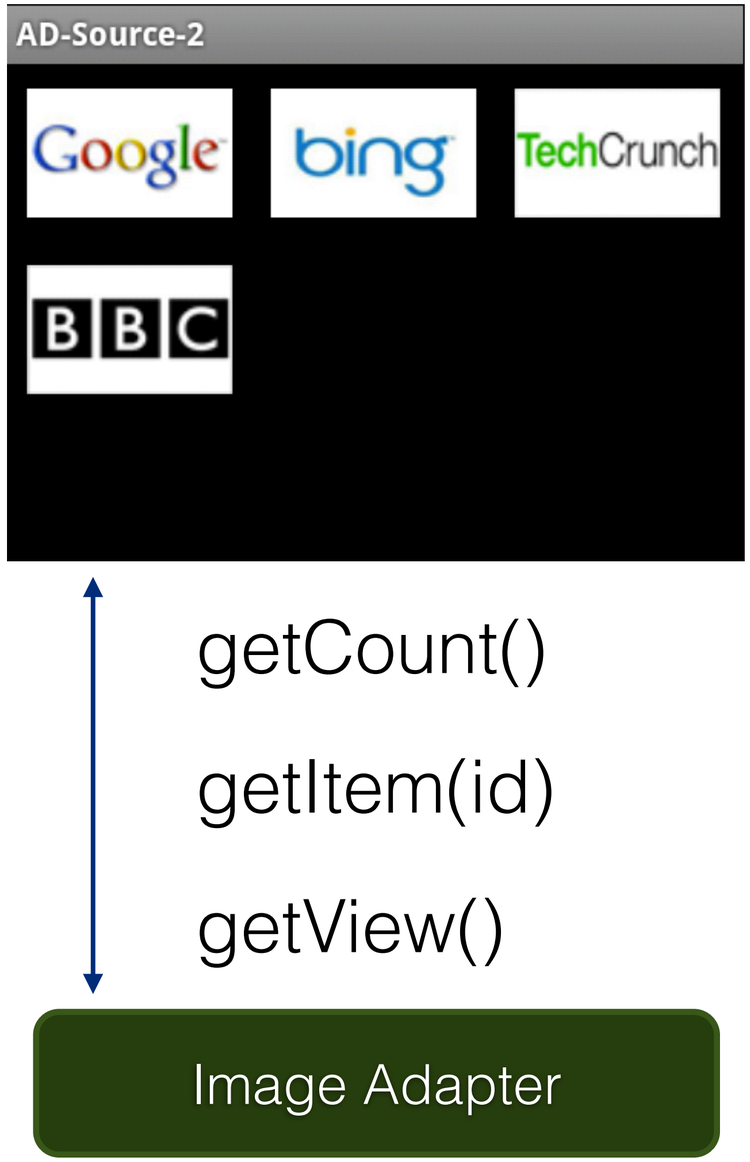
```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

```
@Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.grid_view);

       GridView gridview = (GridView) findViewById(R.id.gridview);
       gridview.setAdapter(new ImageAdapter(this));

       gridview.setOnItemClickListener(new OnItemClickListener() {
         @Override
           public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
               Log.d(TAG, "" + position);
           }
       });
   }
```



Grid View

```
public class ImageAdapter extends BaseAdapter {

    public ImageAdapter(Context c) {
        mContext = c;
    }

    @Override
    public int getCount() { return mThumbIds.length; }

    @Override
    public Object getItem(int position) { return null; }

    public long getItemId(int position) { return 0; }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView imageView;
        if (convertView == null) {
            imageView = new ImageView(mContext);
            ...
        } else {
            imageView = (ImageView) convertView;
        }

        imageView.setImageResource(mThumbIds[position]);
        return imageView;
    }

    // references to our images
    private Integer[] mThumbIds = { R.drawable.google, R.drawable.bing, R.drawable.tc, .drawable.bbc };
}
```



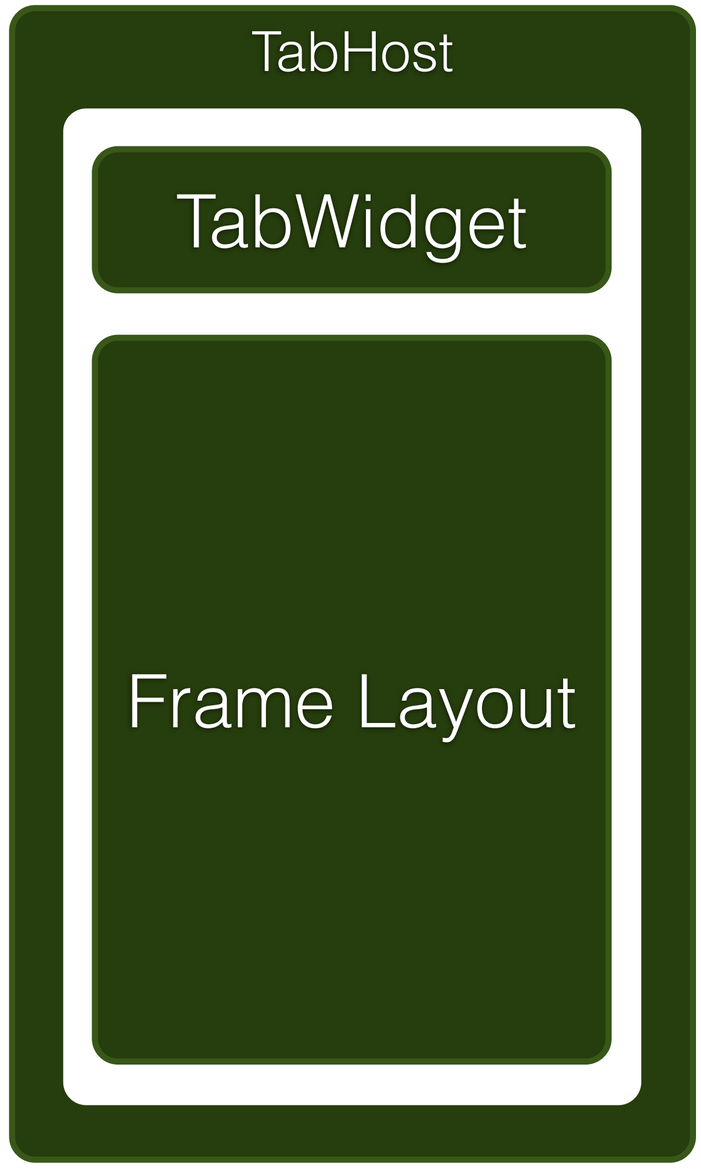
Grid View

- A custom Adapter (ImageAdapter in our case) inherited from BaseAdapter implements some required methods to provide the number of element that should be showed (**getCount()**), the actual object at a specific position (**getItem(int)**) and the row id of the item (**getItemId(int)**).
- The main method is **getView()**. It creates a new View for each element added to the Adapter (Images in our example).
- When this is called, a View is passed in, which is normally a recycled object (at least after this has been called once), so there's a check to see if the object is null. If it is null, an View object (ImageView) is instantiated and configured with desired properties for the image

```
@Override
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView imageView;
        if (convertView == null) {
            imageView = new ImageView(mContext);
            ...
        } else {
            imageView = (ImageView) convertView;
        }

        imageView.setImageResource(mThumbIds[position]);
        return imageView;
    }
```

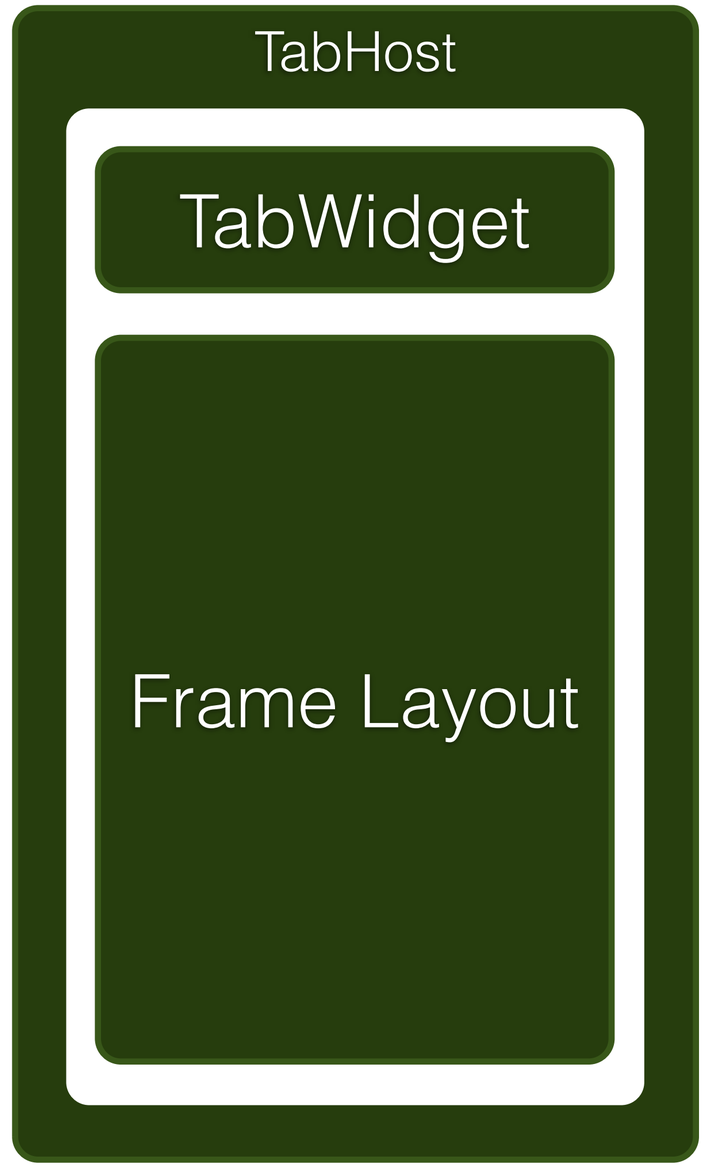
Tab Layout

- 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.
- Tab content could be implemented in two different ways:
  - use the tabs to swap Views within the same Activity
  - use the tabs to change between entirely separate activities.
- The right method depend on your application specs, but if each tab provides a distinct user activity, then it probably makes sense to use a separate Activity for each tab, so that you can better manage the application in discrete groups, rather than one massive application and layout.



Tab Layout

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="5dp">
        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" />
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:padding="5dp" />
    </LinearLayout>
</TabHost>
```



Tab Layout

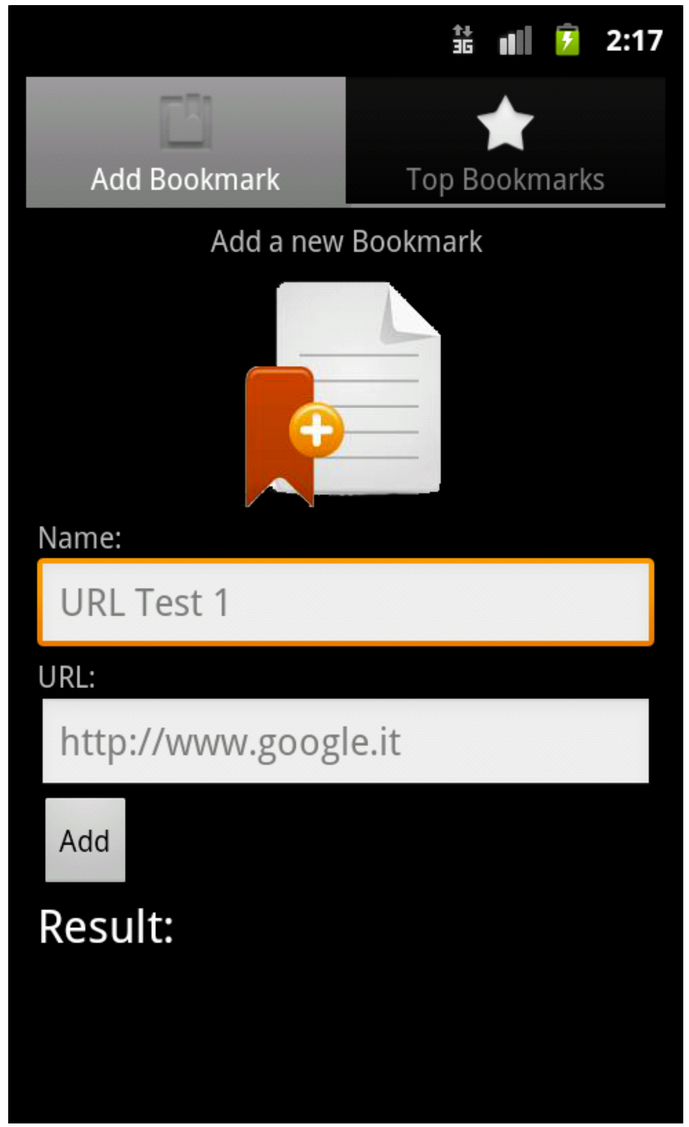
- The main activity associated to the Tab extends “**TabActivity**” and in the onCreate method defines views, icons and characteristics of the layout.

```
Resources res = getResources(); // Resource object to get Drawables
TabHost tabHost = getTabHost();  // The activity TabHost
TabHost.TabSpec spec;  // Resusable TabSpec for each tab
Intent intent;  // Reusable Intent for each tab
```

Init

```
// Create an Intent to launch an Activity for the tab (to be reused)
intent = new Intent().setClass(this, AddBookmarkActivity.class);

// Initialize a TabSpec for each tab and add it to the TabHost
spec = tabHost.newTabSpec("add_bookmark").setIndicator("Add Bookmark",
res.getDrawable(R.drawable.ic_tab_add)).setContent(intent);

tabHost.addTab(spec);
```

Add Activity



- Each tab element is associated to a Drawable resource defining the icon of the tab when it is selected or not.

Tab Bar Selector

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- When selected, use grey -->
    <item android:drawable="@drawable/ic_tab_bookmark_selected"
          android:state_selected="true" />
    <!-- When not selected, use white-->
    <item android:drawable="@drawable/ic_tab_bookmark_unselected" />
</selector>
```

- This is a state-list drawable,applied to tab image. When the tab state changes, the tab icon will automatically switch between the images defined here.

- Generally StateListDrawable is a drawable object defined in XML that uses different images to represent the same graphic. Images change according to the state of the object.

- In a Button for example exist several different states (pressed, focused, or neither) and, using a state list drawable it is possible to provide a different background image for each state.

- You can describe the state list in an XML file. Each graphic is represented by an <item> element inside a single <selector> element. Each <item> uses various attributes to describe the state in which it should be used as the graphic for the drawable.

- During each state change, the state list is traversed top to bottom and the first item that matches the current state is used—the selection is not based on the "best match," but simply the first item that meets the minimum criteria of the state.

Android Application Icons

Icon Design

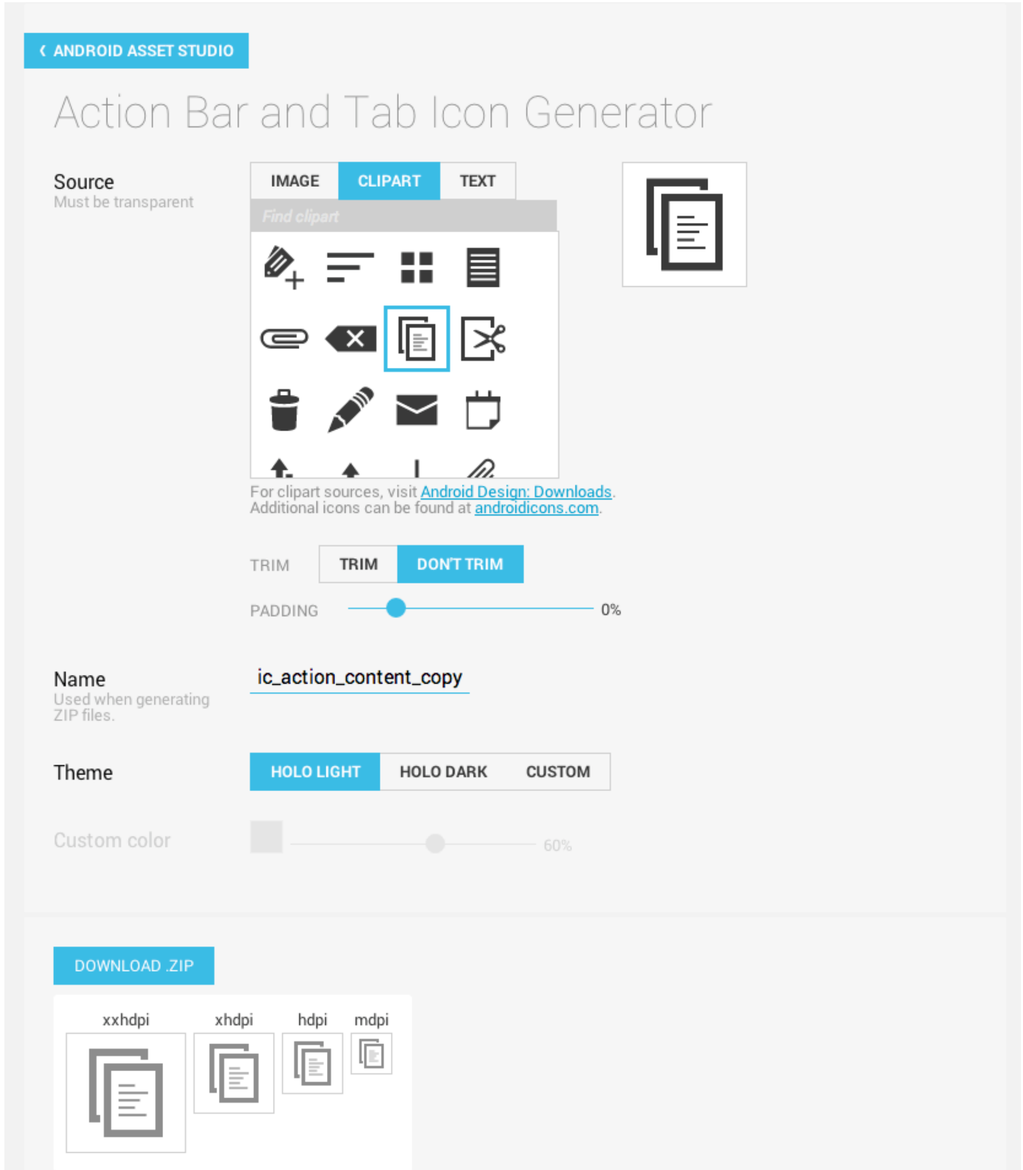
http://developer.android.com/guide/practices/ui_guidelines/icon_design.html

Tab Icon Design

http://developer.android.com/guide/practices/ui_guidelines/icon_design_tab.html

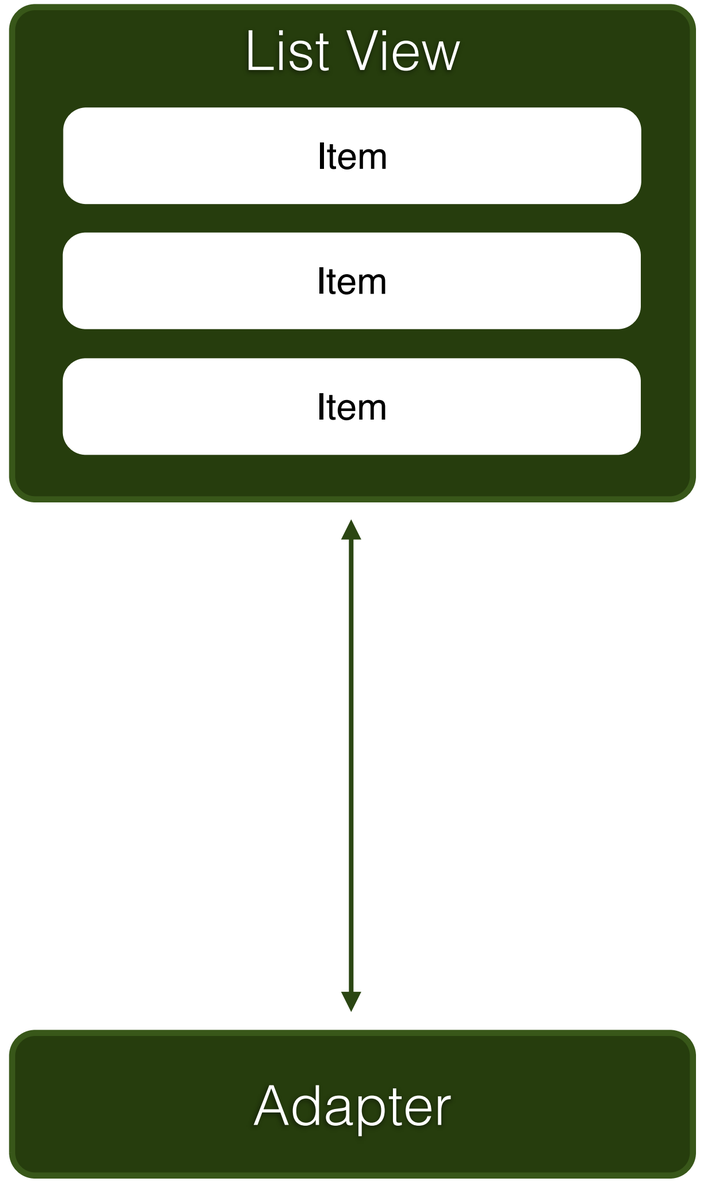
Android Icon Web Tool

http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html



List View

- ListView is a ViewGroup that creates a list of scrollable items. The list items are automatically inserted to the list using a ListAdapter.
- The ListAdapter extends Adapter creating the bridge between a ListView and the data that backs the list.
- The ListView can display any data provided that it is wrapped in a ListAdapter.
- You can use default Adapters like the ArrayAdapter that use a List of String object as input and by default expects that the provided resource id references a single TextView.
- You can create your own Adapter to create customized ListViews according to a specific application requirements.



ListActivity

- An activity that displays a list of items by binding to a data source such as an array or Cursor, and exposes event handlers when the user selects an item.
- ListActivity hosts a ListView object that can be bound to different data sources, typically either an array or a Cursor holding query results.
- ListActivity has a default layout that consists of a single, full-screen list in the center of the screen.
- You can customize the screen layout by setting your own view layout with setContentView() in onCreate(). To do this, your own view **MUST** contain a ListView object with the id "@android:id/list" (or list if it's in code)
- In any case a ListView could be managed also using the traditional Activity class as we will do for our examples. ListActivity is an abstraction to simplify the creation of standard ListView.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingLeft="8dp"
        android:paddingRight="8dp">

    <ListView android:id="@android:id/list"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:background="#00FF00"
              android:layout_weight="1"
              android:drawSelectorOnTop="false"/>

    <TextView android:id="@android:id/empty"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:background="#FF0000"
              android:text="No data"/>
</LinearLayout>
```

Activity & List View

- Using the standard Activity class you can retrieve the ListView object from the associated xml file (through findViewById(...) method) as for other View elements and set:
  - List Adapter (setAdapter(...)) with
    - List contents
    - Element xml descriptor
  - List properties
  - Listeners for List events

Activity Layout File (default_list_view.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
    </ListView>

</LinearLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/
android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:textSize="16sp" >
</TextView>
```

List View Element File (default_list_item.xml)

Activity & List View

Activity Layout File (default_list_view.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
    </ListView>

</LinearLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/
android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:textSize="16sp" >
</TextView>
```

List View Element File (default_list_item.xml)

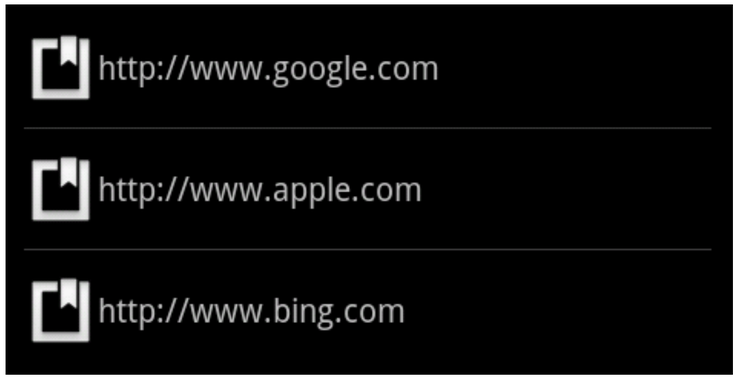
```
public class BookmarkListActivity extends Activity{

    static final String[] BOOKMARKS = new String[] {
        "http://www.google.com", "http://www.apple.com", "http://www.bing.com"
      };

     @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.default_list_view);

           ListView listView = (ListView)findViewById(R.id.listView);
           listView.setAdapter(new ArrayAdapter<String>(this, R.layout.default_list_element, BOOKMARKS));

           listView.setTextFilterEnabled(true);
```

Get the ListView reference and set the Adapter with context, element xml and data array

```
listView.setOnItemClickListener(new OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view,int position, long id) {
        ...
    }

 });
        }
}
```

Set the event listener

Custom ListView Adapter




```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    android:id="@+id/linearLayout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingTop="10dp"
    android:paddingBottom="10dp">
    <ImageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:src="@drawable/bookmark_small_icon" />
    <TextView
        android:id="@+id/bookmarkElementTextView"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:gravity="center_vertical"
    />
</LinearLayout>
```

- You can create your own ArrayAdapter and define a custom xml view for each ListView Element.
- The xml has the same structure of other layout files and can contain traditional Android UI components.

Custom ListView Adapter

Extends **ArrayAdapter** class and override constructor and **getView** method.

In the constructor saves the context reference and the list of objects (String in the example) in order to make them available in the adapter.

The overridden method **getView** it is called bye the OS to instantiate a new View for each element of the ListView. Using a LayoutInflater it is possible to load an Android layout starting from an XML resource and directly work with it to properly configure the view and return it.

The developer should check if the view (convertView) is != null to avoid unnecessary use of LayoutInflater if the View object is already instantiated and available.

```
public class BookmarkListAdapter extends ArrayAdapter<String>{

    private Context mContext = null;
    private List<String> bookmarkList = null;

    public BookmarkListAdapter(Context context,
           int textViewResourceId, List<String> objects) {
        super(context,textViewResourceId, objects);
        this.mContext  = context;
        this.bookmarkList  = objects;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View v = convertView;

        if (v == null) {
            LayoutInflater vi =
            (LayoutInflater)mContext.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            v = vi.inflate(R.layout.custom_list_element, null);
        }

        TextView bookmarkTextView = (TextView)v.findViewById(R.id.bookmarkElementTextView);
        bookmarkTextView.setText(this.bookmarkList.get(position));

        return v;
    }

}
```

Layout Inflater

```
LayoutInflater vi = (LayoutInflater)mContext.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
v = vi.inflate(R.layout.custom_list_element, null);
```

- Instantiates a layout XML file into its corresponding View objects.
- It is never used directly. Instead, use getLayoutInflater() or getSystemService(String) to retrieve a standard LayoutInflater instance that is already hooked up to the current context and correctly configured for the device you are running on.
- It is used to easily and dynamically add views using Java code starting from an XML resource such as in ListView and TableView.
- For performance reasons, view inflation relies heavily on pre-processing of XML files that is done at build time. Therefore, it is not currently possible to use LayoutInflater with an XmlPullParser over a plain XML file at runtime; it only works with an XmlPullParser returned from a compiled resource (R.something file.)

Custom ArrayAdapter

Extends **ArrayAdapter** class with a custom object (BookmarkDescriptor).

In the constructor uses the base class constructor without specifying the object List that by default is a List of Strings.

Since the list is not made by String objects it is necessary to override getCount()
(In the ArrayAdapter<String> this method and the count are provided by the base class) method in order to return to the OS the number of elements filling the list. (It is the same approach user)

```
public class BookmarkListAdapter extends ArrayAdapter<BookmarkDescriptor>{

    ...

    public BookmarkListAdapter(Context context,
            int textViewResourceId, List<BookmarkDescriptor> objects) {
        super(context,textViewResourceId);
        ...
    }

    @Override
    public int getCount() {
        return this.bookmarkList.size();
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View v = convertView;

        if (v == null) {
            LayoutInflater vi =
            (LayoutInflater)mContext.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            v = vi.inflate(R.layout.custom_list_element, null);
        }
        ...
        return v;
    }

}
```

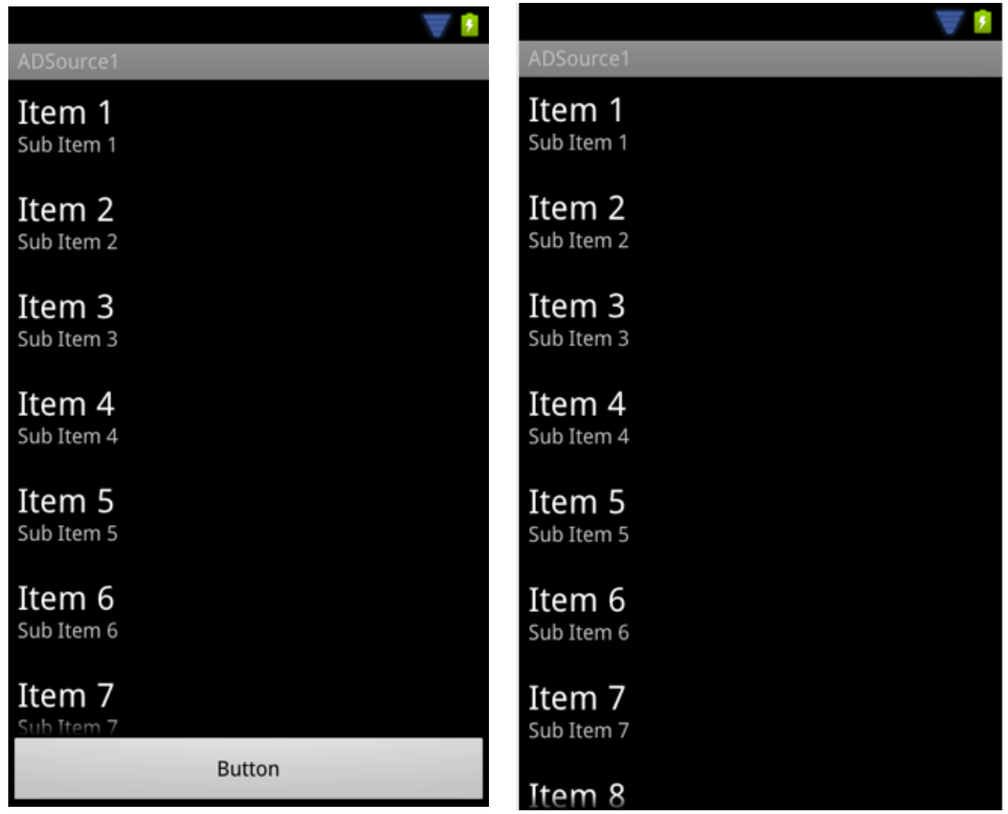
ListView Events

```
listView.setOnItemClickListener(new OnItemClickListener() {
          @Override
          public void onItemClick(AdapterView<?> parent, View view,int position, long id) {
              ...
          }

       });

listView.setOnItemLongClickListener(new OnItemLongClickListener() {

          @Override
          public boolean onItemLongClick(AdapterView<?> parent, View view,int position, long id) {
              ...
              return false;
          }

       });
```

- ListView allows to define the callback for different events. In particular it provides methods to specify a Listener for events triggered when a user interact with an element in the list. These callbacks have the references to the main List, the element View, position and id of the element. Two useful interface definition are:
  - **OnItemClickListener:** callback invoked when an item in this AdapterView has been clicked.
  - **OnItemClickListener:** callback invoked when an item in this view has been clicked and held.

ListView Hint

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

    <ListView android:id="@+id/list"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1.0" />

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button" />

</LinearLayout>
```



- In order to put one ore more Views (like a Button in the example) after a ListView you should remember to set the android:layout_weight to 1.0 leaving the space for additional components.

ScrollView

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

    <LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >
        ...
    </LinearLayout>
</ScrollView>
```

- Layout container for a view hierarchy that can be scrolled by the user, allowing it to be larger than the physical display.
- A ScrollView is a FrameLayout, meaning you should place one child in it containing the entire contents to scroll; this child may itself be a layout manager with a complex hierarchy of objects. A child that is often used is a LinearLayout in a vertical orientation, presenting a vertical array of top-level items that the user can scroll through.
- The TextView class also takes care of its own scrolling, so does not require a ScrollView, but using the two together is possible to achieve the effect of a text view within a larger container.
- ScrollView only supports vertical scrolling.

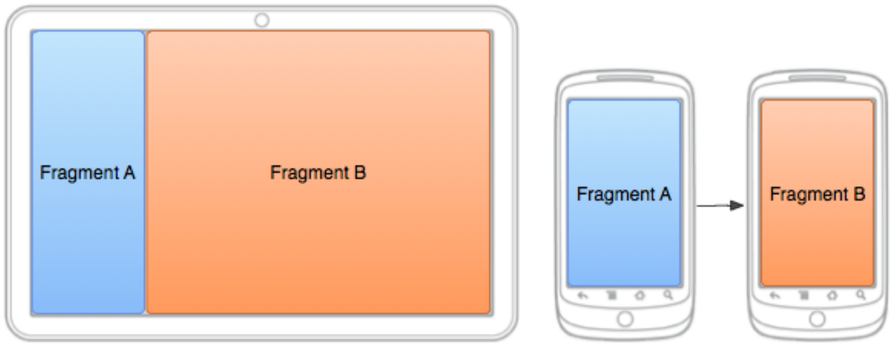
Start Activity with Parameters

```
Bundle bundle = new Bundle();
bundle.putInt("bookmarkPosition", position);

Intent newIntent = new Intent(getApplicationContext(), EditBookmarkActivity.class);
newIntent.putExtras(bundle);
startActivityForResult(newIntent);
```

- Using the method startActivity(Intent) or startActivityForResult(Intent, result) you can start a new activity, which will be placed at the top of the activity stack.
- With the second version you can specify an additional integer parameter identifying the call. The result will come back through your **onActivityResult(int, int, Intent)** method.
- Th Bundle class is a map **Key<->Value** that allows to specify parameters associated to an Intent (through the method **putExtras(bundle)**) that you want to send to the new Activity.
- In the target Activity you can retrieve the incoming Bundle and extract sent values using the right Key.

```
Bundle bundle = this.getIntent().getExtras();
bookmarkPosition = bundle.getInt("bookmarkPosition");
```

Fragments

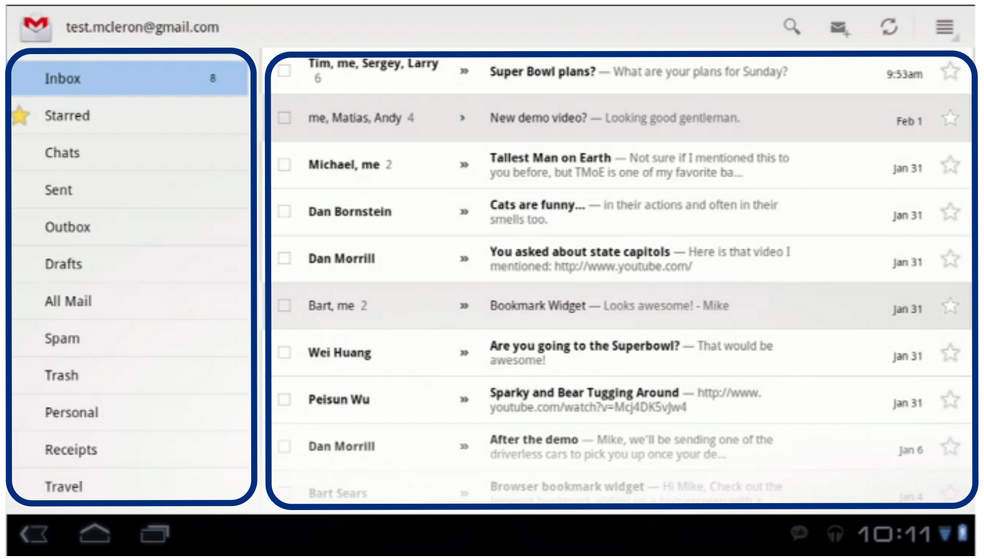


- A Fragment is a portion of the user interface in an Activity with a specific and potentially different behavior.
- It is possible to combine multiple fragments in a single activity in order to create a UI with multiple panels and reuse a fragment in multiple activities.
- Essentially a Fragment can be viewed as a modular section of an activity that
  - has its own lifecycle
  - receives its own input events
  - which can be added or removed while the activity is running

http://developer.android.com/guide/components/fragments.html

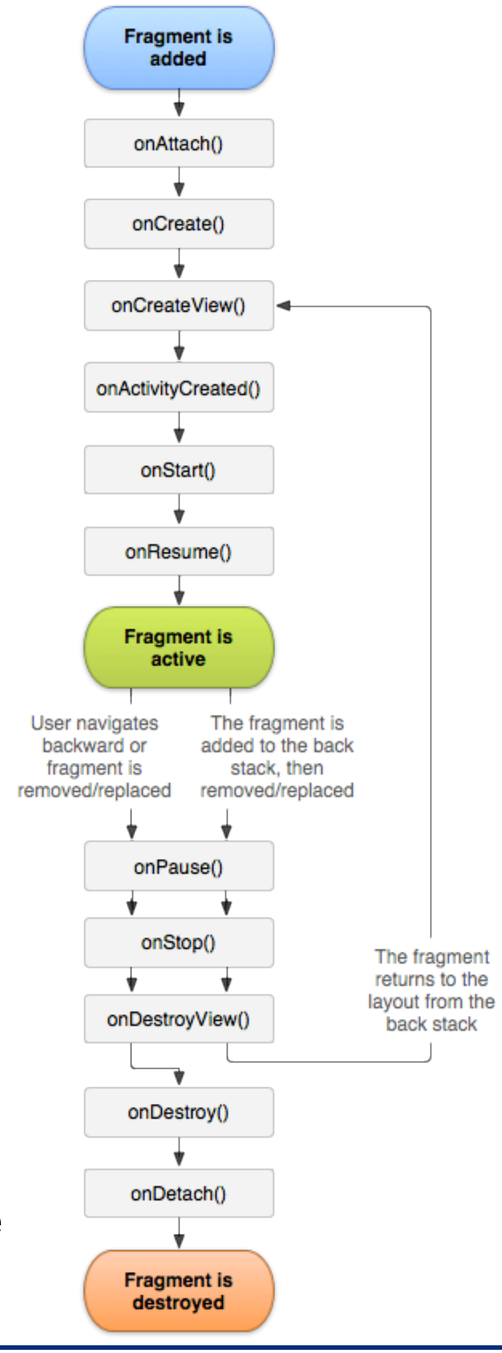
Fragments Philosophy

- **E.g:**
  - an email application can use one fragment to show the list of received e-mails on the left and another fragment to display the selected e-mail on the right
  - both fragments appear in one activity, side by side
  - each fragment has its own set of lifecycle callback methods and handle their own user input events.
  - Using the fragments the user can in the same activity refresh the list of e-mails while he/she is reading an already selected message without changing the UI.



Fragment LifeCycle

- To create a fragment, you must create a subclass of Fragment (or an existing subclass of it).
- The Fragment class looks appears like an Activity. It contains callback methods similar to an activity, such as onCreate(), onStart(),onPause(), and onStop().
- You can implement at least the following lifecycle methods to define your behavior:
  - onCreate(): The system calls this when creating the fragment. You should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.
  - onCreateView(): Called by the System when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a View from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.
  - onPause(): Method called by the System as a first feedback that the user is leaving the fragment (though it does not always mean the fragment is being destroyed). This is usually where you should commit any changes that should be persisted beyond the current user session.



Fragment Code

To define a layout for a fragment, you must implement the onCreateView() callback method called by the system when it's time for the fragment to draw its layout.

In order to return a Layout/View from the method onCreateView(), you can inflate it from a layout resource defined in XML. To help you the method provides a LayoutInflater object already created.

```
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
    {
        // Inflate the layout for this fragment
        View view = inflater.inflate(R.layout.example_fragment, container, false);
        return view;
    }
}
```

The container parameter represents the parent ViewGroup (from the activity's layout) in which the fragment layout will be inserted.

In order to return a Layout/View from the method onCreateView(), you can inflate it from a layout resource defined in XML. To help you the method provides a LayoutInflater object already created.

Adding a fragment to an activity

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.ArticleListFragment"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
            android:id="@+id/viewer"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

Adding a fragment to an activity

While your activity is running, you can add one or more fragments to your existing activity layout. You simply need to specify a ViewGroup in which to place the fragment. The management and the interaction between an Activity and a Fragment is defined and controlled by the class FragmentManager.

To make fragment transactions in your activity (add, remove, or replace a fragment), you must use APIs from FragmentTransaction.

```
FragmentManager fragmentManager = getFragmentManager()
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();

ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

You can then add a fragment using the add() method, specifying the fragment to add and the view in which to insert it. The first argument passed to add() is the ViewGroup in which the fragment should be placed, specified by resource ID, and the second parameter is the fragment to add. Once you've made your changes with FragmentTransaction, you must call commit() for the changes to take effect.

Application communication with Fragments

- Although a Fragment is implemented as an object that is independent from an Activity and can be used inside multiple activities, a given instance of a fragment is directly tied to the activity that contains it.
- Specifically, the fragment can access the Activity instance with getActivity() and easily perform tasks such as find a view in the activity layout.

```
View listView = getActivity().findViewById(R.id.list);
```

- At the same an activity can call methods in the fragment by acquiring a reference to theFragment from FragmentManager, using findFragmentById() or findFragmentByTag().

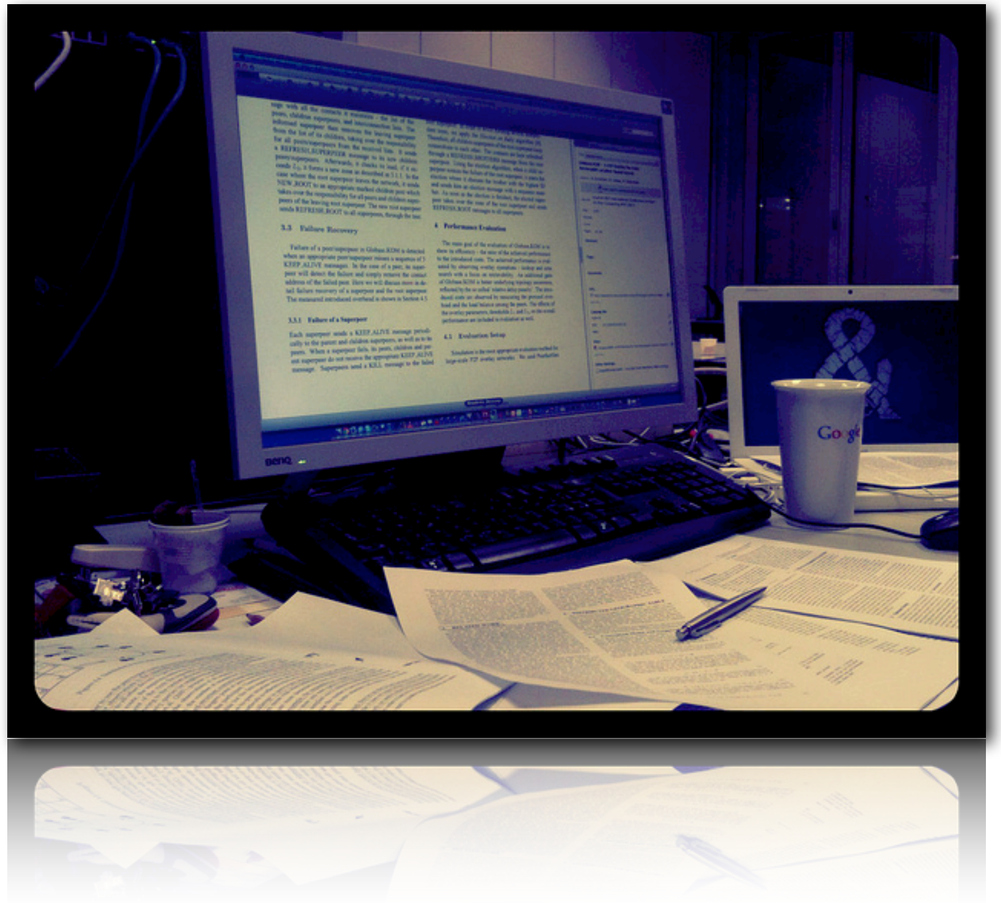
```
ExampleFragment fragment = (ExampleFragment)
getFragmentManager().findFragmentById(R.id.example_fragment);
```

Fragments & Android Compatibility

- Android 3.0, or Honeycomb, came with some fundamental user interface changes, most notably in the form of the Fragment API.
- In order to support previous Android versions Google has released a library called the Android Compatibility package. (You should download and add the library to your project)
- This package provides support for the Fragment API as well as other key new features to devices as far back as Android 1.6.
- To support Fragment is previous version you should
  - Change the Activity to extend the FragmentActivity class when you want to use them
  - Change the call to the getFragmentManager() method to the getSupportFragmentManager() method
  - Organize or update the imports and the code (import android.support.v4.app.Fragment;) (android.support.v4.app.FragmentTabHost)

Coming Up

- Next Lecture
  - Android Graphical User Interface - 2
- Homework
  - Review Bookmark Application



Android Development

Lecture 3
Android Graphical User Interface 1