UI - Layout

Sang Shin
Michèle Garoche
www.javapassion.com
“Learn with Passion!”

Disclaimer

- Portions of this presentation are modifications based on work created and shared by the Android Open Source Project
  - http://code.google.com/policies.html
- They are used according to terms described in the Creative Commons 2.5 Attribution License
  - http://creativecommons.org/licenses/by/2.5/

Topics

- What is Layout & How to declare Layout
- Layout File structure and location
- Attributes
  - ID
  - Layout parameters
- Types of Layout
  - LinearLayout
  - RelativeLayout
  - TableLayout
  - FrameLayout
  - Tab layout

What is a Layout &
How to declare a Layout

What is a Layout?

- It defines the layout structure for the user interface in an Activity
- It holds all the visual elements that appear to the user

How to declare a Layout? Two Options

- Option #1: Declare layout and its visual elements in XML (most common and preferred)

- Option #2: Instantiate a layout and its visual elements at runtime (programmatically in Java code)

Using both options

- You can use either or both of these options for declaring and managing your application's UI (layout + visual elements)
- Android system create Java objects for the visual elements defined in the XML
  - Every View and ViewGroup has a corresponding Java class (That is the reason why we use the terms “View element” and “View class” interchangeably)
- Example usage scenario of using both
  - You could declare your application's default layouts in XML, including the visual elements and their properties. (Option #1)
  - You could then add code in your application that would modify the properties of the visual elements

Advantages of Option #1: Declaring UI in XML

- Separation of the presentation (UI) from the code that controls its behavior
  - You can modify UI without having to modify your source code and recompile
  - For example, you can create XML layouts for different screen orientations, different device screen sizes, and different languages
- Easier to visualize the structure of your UI (without writing any code)
  - Easier to design/debug UI
  - Visualizer tool (like the one in Eclipse IDE)

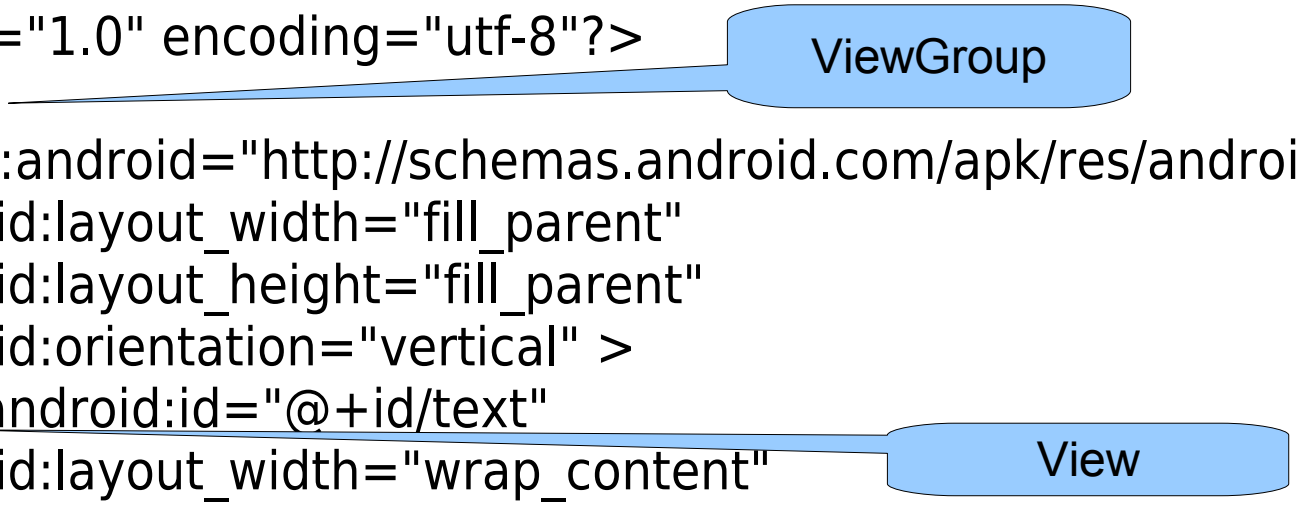
Layout File

Layout File Structure

- A layout specifies a hierarchical tree structure of ViewGroup and View elements
  - A ViewGroup is considered as a branch
  - A View is considered as leaf
- A layout file must contain exactly one root element, which must be one of the following
  - ViewGroup element (i.e., LinearLayout) - typical
  - View element (Button, for example)
- A ViewGroup element can have child elements, which themselves can be ViewGroup or View elements

Example: Layout File

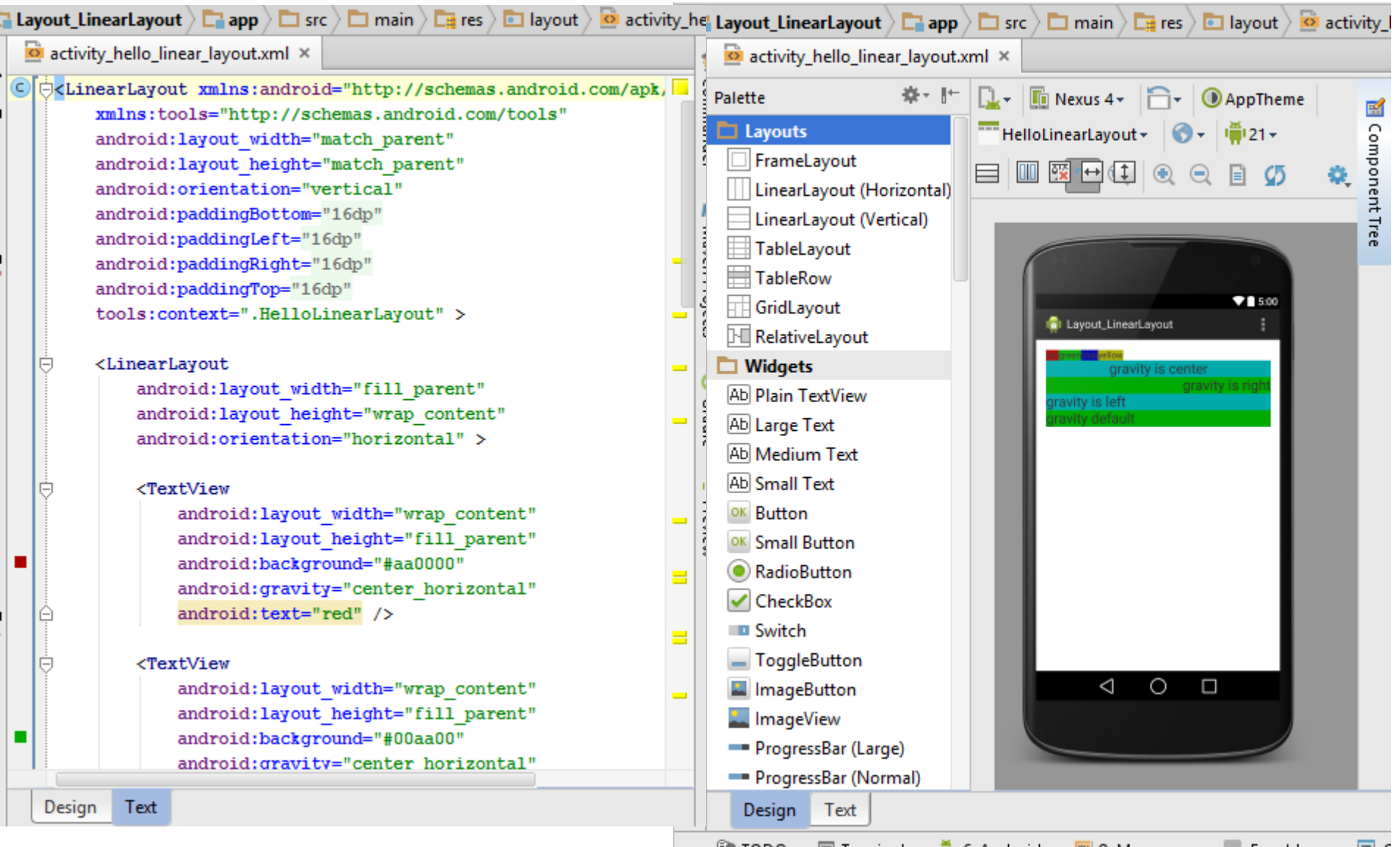
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >
   <TextView android:id="@+id/text"
        android:layout_width="wrap_content"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



Where to create Layout file?

- Save the file with the .xml extension, in your Android project's *res/layout/* directory
- Update using xml or visual design layout

Layout design from xml or visual



Load the Layout XML Resource

- Each XML layout file is compiled into a layout resource
  - A layout resource is referred to as *R.layout.<layout_file_name>*
- The layout resource is loaded through *setContentView(R.layout.<layout_file_name>)*

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

Attributes

Attributes

- Every View and ViewGroup element has a set of attributes.
  - Some attributes are specific to a specific View element (for example, the *textSize* attribute is only relevent to *TextView*)
  - Some are common to all View elements, because they are inherited from the root View element (like the *id* attribute).
- These attributes are typically in XML form but can be set programmatically

Attributes: ID

ID Attribute

- XML attribute common to all View objects (defined by the View class)
  - A View element has 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
- Syntax – *android:id="@+id/my_button"*
  - @ indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.
  - + means that this is a new resource name that must be created and added to our resources (in the R.java file)

Example: ID Attribute

```
<?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/my_text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
   <Button android:id="@+id/my_button"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Hello, I am a Button" />
</LinearLayout>
```

Example: Generated R.java

```
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class id {
        public static final int my_button=0x7f050001;
        public static final int my_text=0x7f050000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

Android's Built-in Resource ID

- Android framework comes with its own built-in resources
- When referencing an Android's built-in resource in the layout resource file, you do not need the plus-symbol, but must add the android package namespace
  - *android:id="@android:id/empty"*
- When referencing an Android's built-in resource in the Java code, it is referenced through android package namespace
  - *android.R.id.empty*

How to reference views in Java code?

- Assuming a view/widget is defined in the layout file with a unique ID

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

- Then you can make a reference to the View element via *findViewById(R.id.<string-id>)*.

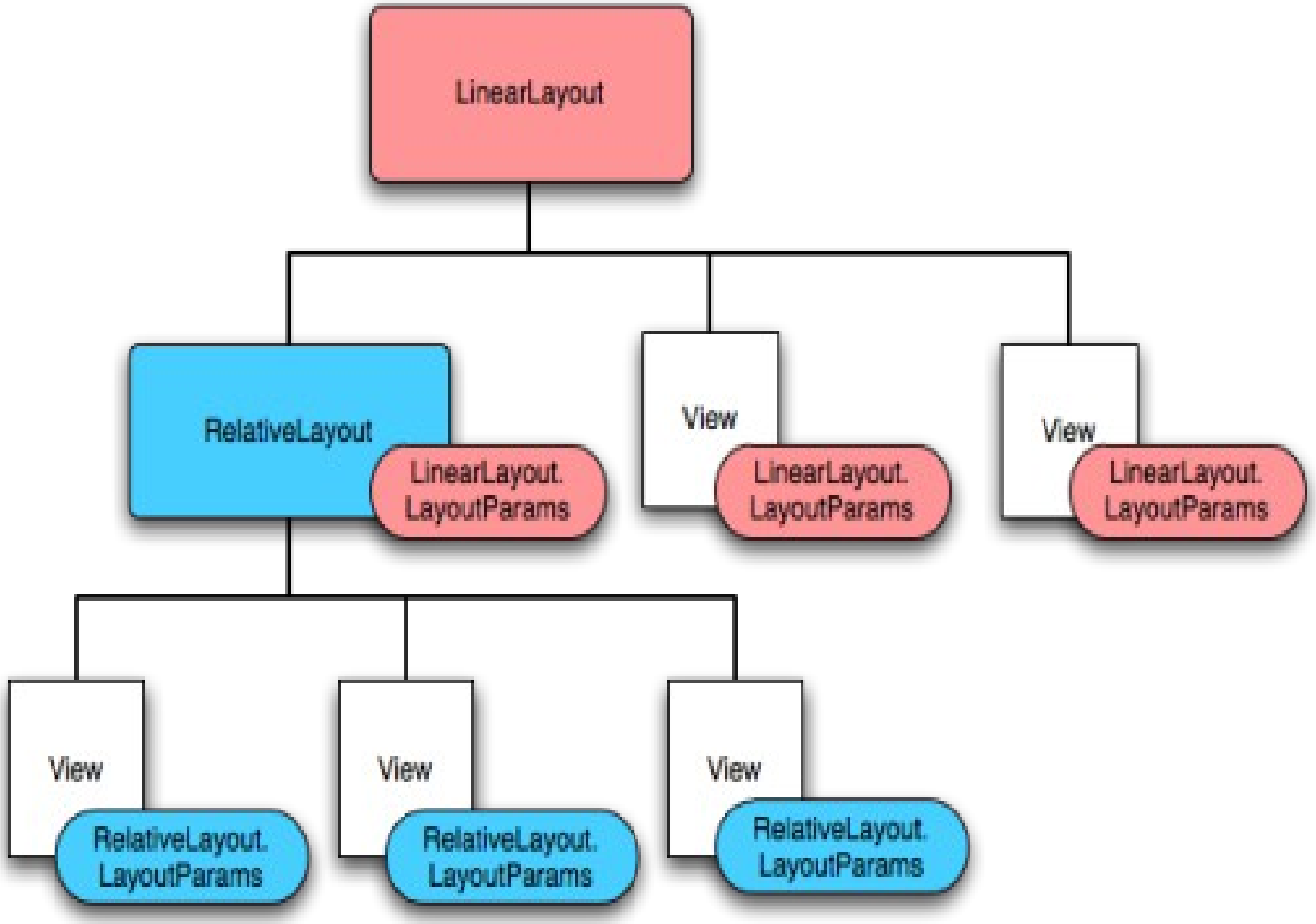
```
// The value of R.id.my_button is defined in the
// mypackage.R.java
Button myButton = (Button) findViewById(R.id.my_button);
```

Attributes:
Layout Parameters

What Are Layout Parameters?

- Layout parameters are special attributes that describe certain layout orientations of a View element, as defined by that object's parent ViewGroup object.
- Named as *layout_<something>*
  - *layout_width*
  - *layout_height*
  - *layout_weight*
  - *layout_gravity*
  - *...*

Parent view group defines layout parameters for each child view (including the child view group)



Values of *layout_width* & *layout_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.
- *match_parent*
  - Same as *fill_parent*
  - Introduced in API Level 8

```
<Button android:id="@+id/my_button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

layout_weight attribute

- Is used in a LinearLayout to assign "importance" to child Views within that layout.
- All Views have a default layout_weight of value 0
  - They take up only as much room on the screen as they need to be displayed
- Assigning a value higher than 0 will split up the rest of the available space in the parent View
  - <each View's layout_weight>/<total value of layout_weight of all View's>

Layout Types

Layout Types

- All layout types are subclass of *ViewGroup* class
- Layout types
  - *LinearLayout*
  - *RelativeLayout*
  - *TableLayout*
  - *FrameLayout*
  - Tab layout

LinearLayout

- Aligns all children in a single direction — vertically or horizontally, depending on how you define the *orientation* attribute.
- All children are stacked one after the other, so a vertical list will only have one child per row
- A LinearLayout supports
  - margins between children
  - gravity (right, center, or left alignment) of each child.
  - weight to each child

RelativeLayout

- RelativeLayout lets child views specify their position relative to the parent view or to each other (specified by ID)
  - You can align two elements by right border, or make one below another, centered in the screen, centered left, and so on
- Elements are rendered in the order given
  - If the first element is centered in the screen, other elements aligning themselves to that element will be aligned relative to screen center.

RelativeLayout Example

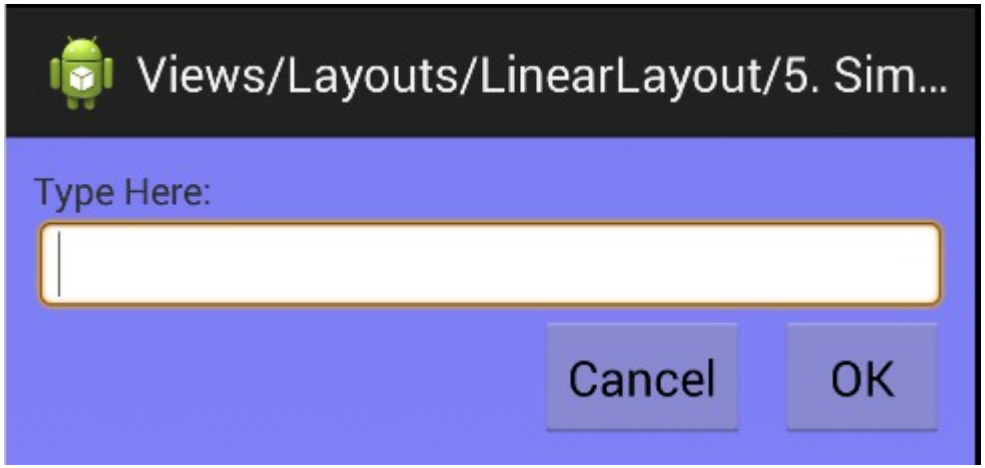
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:background="@drawable/blue"
            android:padding="10px" >

   <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="10px"
        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>
```



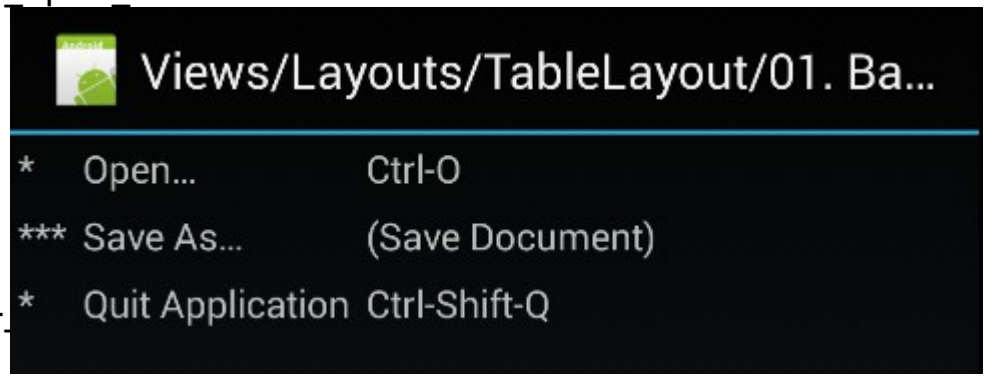
TableLayout

- TableLayout positions its children into rows and columns
- *TableRow* objects are the child views of a TableLayout
  - Each TableRow defines a single row in the table
  - Each row has zero or more cells, each of which is defined by any kind of other View.
- Columns can be
  - Hidden
  - Stretch and fill the available screen space
  - Shrinkable to force the column to shrink until the table fits the screen

TableLayout Example

```
<?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:text="@string/table_layout_4_open"
         android:padding="3dip" />
      <TextView
         android:text="@string/table_layout_4_open_shortcut"
         android:gravity="right"
         android:padding="3dip" />
   </TableRow>

   <TableRow>
      <TextView
         android:text="@string/table_layout_4_
         android:padding="3dip" />
      <TextView
         android:text="@string/table_layout_4_save_shortcut"
         android:gravity="right"
         android:padding="3dip" />
   </TableRow>
</TableLayout>
```



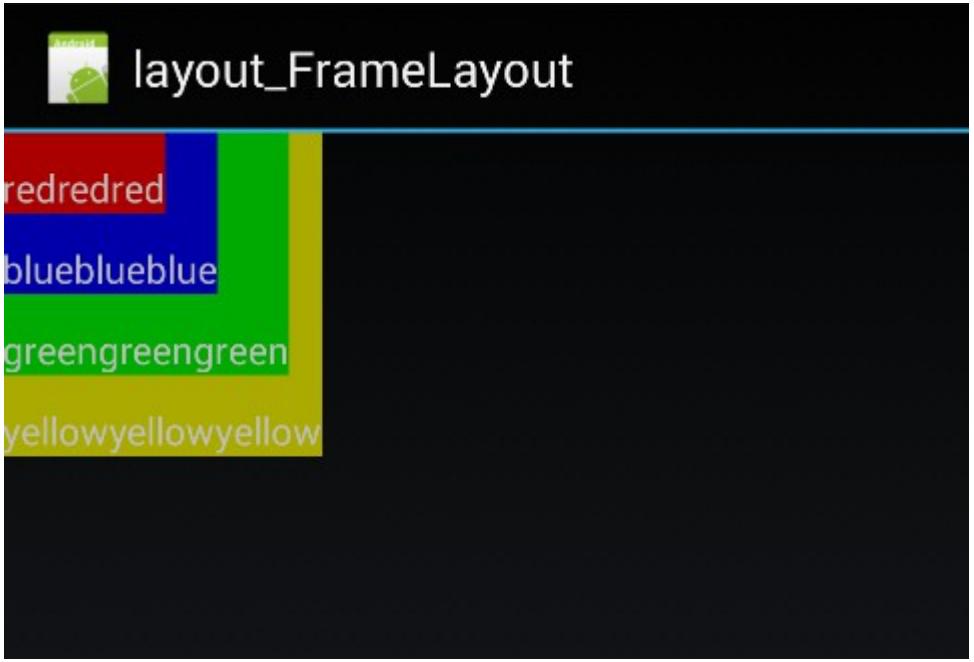
FrameLayout

- FrameLayout is the simplest type of layout object.
- It's basically a blank space on your screen that you can later fill with a single object
  - For example, a picture that you'll swap in and out.
- All child elements of the FrameLayout are pinned to the top left corner of the screen; you cannot specify a different location for a child view.
  - Subsequent child views will simply be drawn over previous ones, partially or totally obscuring them (unless the newer object is transparent).

FrameLayout Example

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

     <TextView
        android:text="yellowyellowyellow"
        android:gravity="bottom"
        android:background="#aaaa00"
        android:layout_width="wrap_content"
        android:layout_height="120dip"/>
     <TextView
        android:text="greengreengreen"
        android:gravity="bottom"
        android:background="#00aa00"
        android:layout_width="wrap_content"
        android:layout_height="90dip" />
    <TextView
        android:text="blueblueblue"
        android:gravity="bottom"
        android:background="#0000aa"
        android:layout_width="wrap_content"
        android:layout_height="60dip" />
     <TextView
        android:text="redredred"
        android:gravity="bottom"
        android:background="#aa0000"
        android:layout_width="wrap_content"
        android:layout_height="30dip"/>
</FrameLayout>
```



Tab Layout

Thank you!

Check JavaPassion.com Codecamps!
http://www.javapassion.com/codecamps
"Learn with Passion!"