Layouts: Organizing the Screen

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/android-tutorial/

© 2012 Marty Hall

For live Android training, please see courses at http://courses.coreservlets.com/.

Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this Android tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 6 or 7 programming, custom mix of topics
  - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, etc.) or survey several
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, EJB3, GWT, Hadoop, SOAP-based and RESTful Web Services

Contact hall@coreservlets.com for details

Topics in This Section

- LinearLayout
- Strategy of nesting layouts
- Using color files
  - And preview of Localization
- Layout weights
- RelativeLayout
- TableLayout
- hierarchyviewer

Overview

Main Layout Strategies

- **XML-based**
  - Declare layout in res/layouts/some_layout.xml
    - Set various XML properties
    - Use visual editor in Eclipse
  - Load with setContentView(R.layout.some_layout)
- **Java-based**
  - Instantiate layout, set properties, insert sub-layouts
    - LinearLayout window = new LinearLayout(this);
    - window.setVariousAttributes(…);
    - window.addView(widgetOrLayout);
  - Load with setContentView(window)
- **This tutorial**
  - Uses XML-based approach. However, attributes can be adapted for Java-based approach.

XML Layout Attributes

- **Idea**
  - Each Layout class has an inner class called LayoutParams that defines general XML parameters that layout uses. These parameters are always named android:layout_*blah*, and usually have to do with sizes and margins.
  - Layout classes define more specific attributes. Many inherited from LinearLayout (which extends ViewGroup and View).
    - *Not* named beginning with "layout_"
- **Example**

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal"
    android:background="@color/color_1">…<LinearLayout>
```

Commonly Used Attributes

- **Size**
  - android:layout_height, android:layout_width
    - match_parent: fill the parent space (minus padding)
      - Renamed from fill_parent in older versions
    - wrap_content: use natural size (plus padding)
    - An explicit size with a number and a dimension. See margins on next slide.
  - android:layout_weight
    - A number that gives proportional sizes. See example.
- **Alignment**
  - android:layout_gravity
    - How the View is aligned within containing View.
  - android:gravity
    - How the text or components inside the View are aligned.
  - Possible values
    - top, bottom, left, right, center_vertical, center_horizontal, center (i.e., center both ways), fill_vertical, fill_horizontal, fill (i.e., fill both directions), clip_vertical, clip_horizontal

Commonly Used Attributes (Continued)

- **Margins (blank space outside)**
  - android:layout_marginBottom, android:layout_marginTop, android:layout_marginLeft, android:layout_marginRight
  - Units (e.g., "14.5dp") – *negative values are legal*
    - dp: density-independent pixels (scaled by device resol.)
    - sp: scaled pixels (scaled based on preferred font size)
    - px: pixels
    - in: inches
    - mm: millimeters
- **Padding (blank space inside)**
  - android:paddingBottom, android:paddingTop, android:paddingLeft, android:paddingRight
    - Values are numbers with units as above

Commonly Used Attributes (Continued)

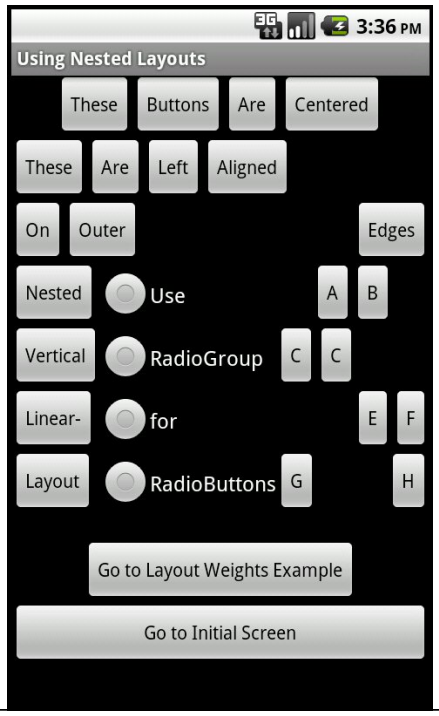
- **ID**
  - android:id
    - Used if the Java code needs a reference to View
    - Used in RelativeLayout so XML can refer to earlier ids
- **Colors**
  - android:background (color or image, for any Layout)
  - android:textColor (e.g., for TextView or Button)
  - Common color value formats
    - "#rrggbb", "#aarrggbb", "@color/color_name"
- **Click handler**
  - android:onClick
    - Should be a public method in main Activity that takes a View (the thing clicked) as argument and returns void

LinearLayout Basics

LinearLayout

- **Idea**
  - Put components in a single row or single column
  - *By nesting, can have rows within columns, etc.*
- **Most important XML attributes**
  - android:orientation
    - "horizontal" (a row) or "vertical" (a column)
    - horizontal is the default, so can be omitted for rows
  - android:gravity
    - How the Views inside are aligned.
    - Possible values
      - top, bottom, left, right, center_vertical, center_horizontal, center (i.e., center both ways), fill_vertical, fill_horizontal, fill (i.e., fill both directions), clip_vertical, clip_horizontal

Example Summary (Highly Nested Layouts)



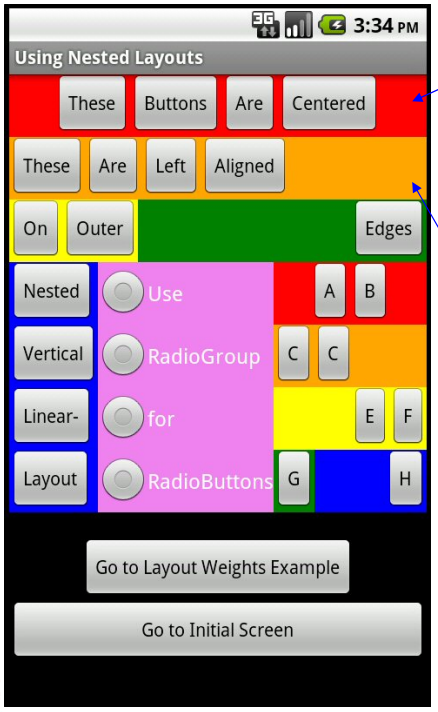
- **General Approach**

```
<?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">

<!-- Widgets and nested layouts -->

</LinearLayout>
```

Example Details



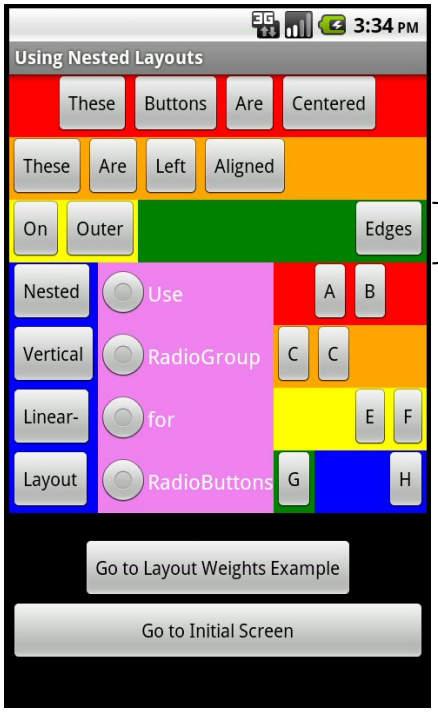
Horizontal LinearLayout with gravity of center_horizontal.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal"
    android:background="@color/color_1">
    <Button android:text="These"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <Button android:text="Buttons"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <Button android:text="Are"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <Button android:text="Centered"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

Horizontal LinearLayout with gravity of left. Otherwise almost same as first row.

Remember that horizontal is the default for android:orientation, so this attribute was omitted for these two rows. The colors will be explained later in this tutorial.

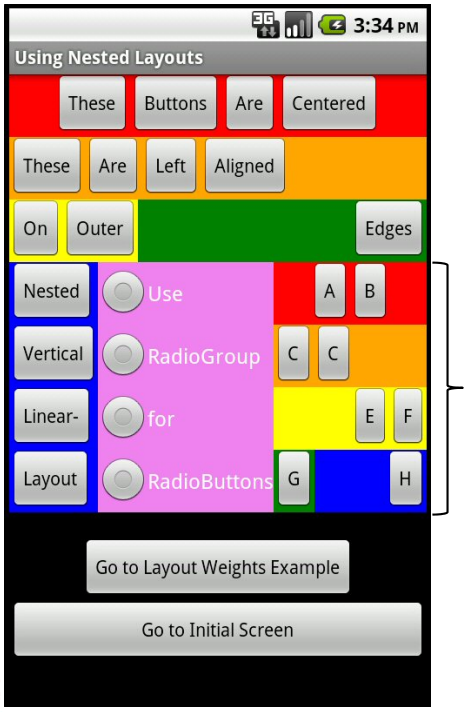
Example Details



Horizontal LinearLayout.

That Layout then contains two more horizontal LinearLayouts. The first (yellow) has android:layout_width of "wrap_content" and android:gravity of "left". The second (green) has android:layout_width of "match_parent" and android:gravity of "right".

Example Details

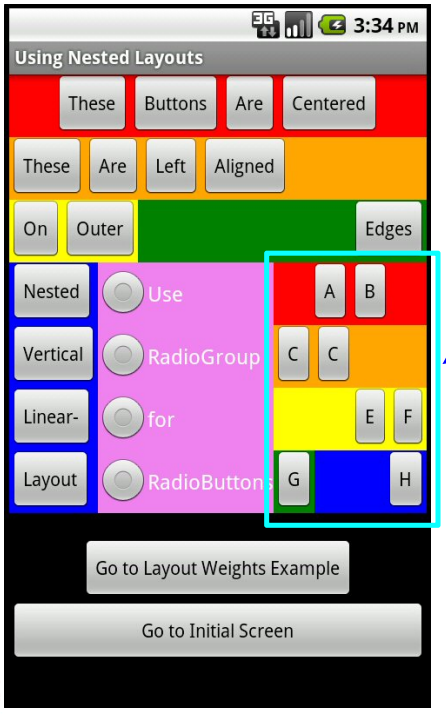


Horizontal LinearLayout.

That Layout then contains three vertical nested layouts. The first (blue) is a LinearLayout with android:orientation of "vertical" and four Buttons inside. The second (violet) is a RadioGroup (similar to LinearLayout but specific to enclosing RadioButtons and making them mutually exclusive), also with android:orientation of "vertical". It has four RadioButtons inside. The third is a LinearLayout with android:orientation of "vertical" and four nested LinearLayouts inside (details on next slide).

The first two columns (nested layouts) have android:layout_width of "wrap_content", and the third has android:layout_width of "match_parent".

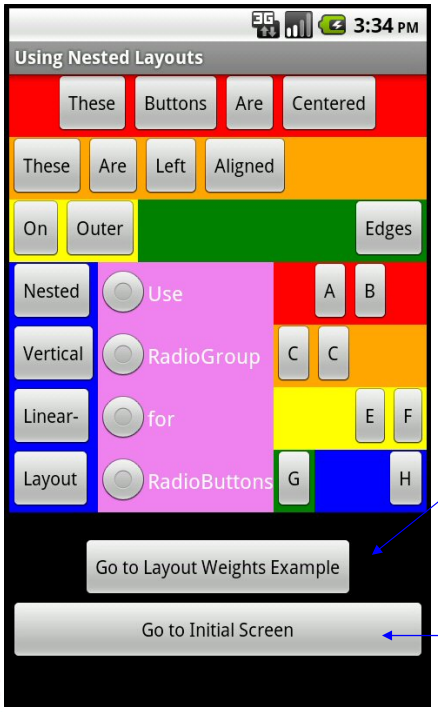
Example Details



Vertical LinearLayout.

That Layout then contains four horizontal nested layouts. The first (red) has android:gravity of "center_horizontal". The second (orange) has android:gravity of "left". The third (yellow) has android:gravity of "right". The fourth contains two further nested horizontal LinearLayouts. The first (green) has android:layout_width of "wrap_content" and android:gravity of "left". The second (blue) has android:layout_width of "match_parent" and android:gravity of "right".

Example Details



Button
android:layout_width="wrap_content"
android:layout_gravity="center_horizontal"
android:layout_marginTop="20dp"

Button
android:layout_width="match_parent"

Setting Colors (& Localization Preview)

Colors

- **Idea**
  - Although colors can be defined explicitly within layout file (e.g., background="#ff0000"), usually more flexible to define color names in separate file, so they can be changed all at once. Refer to color with "@color/color_name".
- **Syntax**

```
<resources>
<color name="color_name_1">#rrggbb</color>
… <!-- Other colors -->
</resources>
```

- **Convention**
  - Use res/values/colors.xml
    - However, any file name is legal. Sometimes it makes more sense to define all attributes (strings, arrays, colors) of a View in a single file dedicated to that view.

Color File (res/values/colors.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="color_1">#ff0000</color>
    <color name="color_2">#ffa500</color>
    <color name="color_3">#ffff00</color>
    <color name="color_4">#008000</color>
    <color name="color_5">#0000ff</color>
    <color name="color_6">#ee82ee</color>
</resources>
```

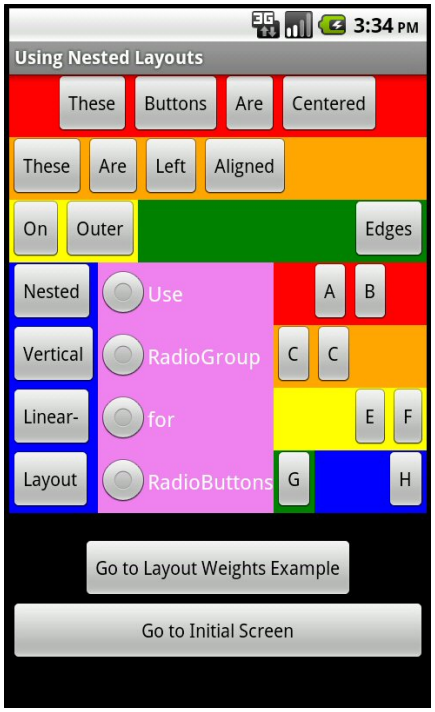
Layout File (res/layouts/nested_layouts.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="…"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:background="@color/color_1">
        ...
     </LinearLayout>

     <!-- All leaf layouts (i.e., ones that don't contain
          nested layouts) given background colors -->

</LinearLayout>
```


Result



Localization Preview

- **Idea**
  - You can store colors or other files in res/values-xy instead of res/values. If the Locale is set to xy, then that file is loaded after the files in res/values.
    - If names match, later file overrides value from earlier file
- **Usual approach**
  - Locale is set for entire phone by end user
- **Approach used here**
  - Locale is set programmatically
- **Many more details**
  - In later lecture on localization

Setting Locale Programmatically

- **Usual purpose**
  - If user sometimes wants to run app in one language and other times in a different language.
    - Again, more common for end user to set Locale for entire phone, not for individual apps.
- **Purpose here**
  - Set the Locale to a fake value ("qq") just so that we can replace colors.xml with another version that makes all the background colors be black.

Setting Locale Programmatically

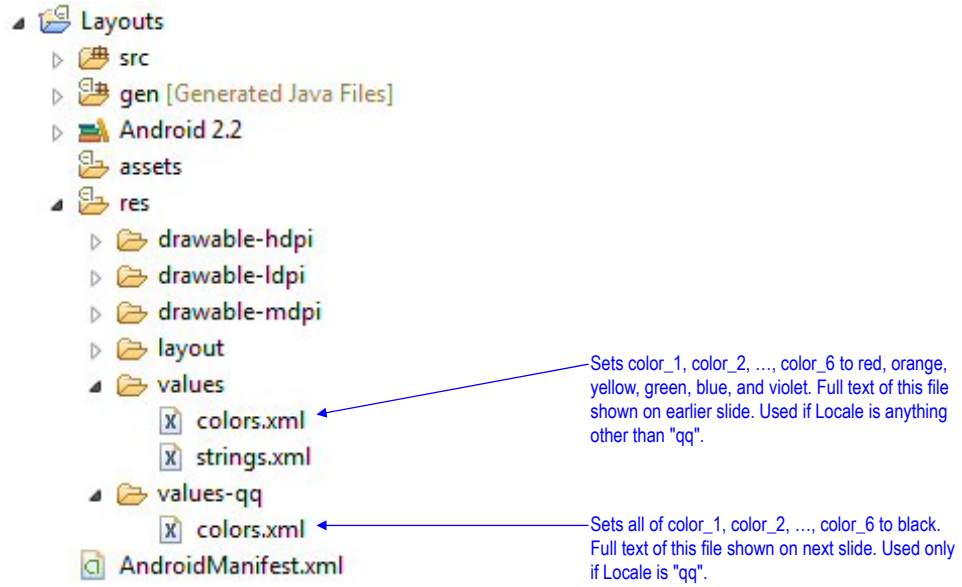
- **Steps**

```
Locale locale = new Locale("es");  // Language code
Locale.setDefault(locale);
Configuration config = new Configuration();
config.locale = locale;
context.getResources().updateConfiguration(config,
                                                null);
```

  - context above is reference to the main Activity

- **More details**
  - http://adrianvintu.com/blogengine/post/Force-Locale-on-Android.aspx

Project Layout



Localized Color File (res/values-qq/colors.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="color_1">#000000</color>
    <color name="color_2">#000000</color>
    <color name="color_3">#000000</color>
    <color name="color_4">#000000</color>
    <color name="color_5">#000000</color>
    <color name="color_6">#000000</color>
</resources>
```


Main Java Code

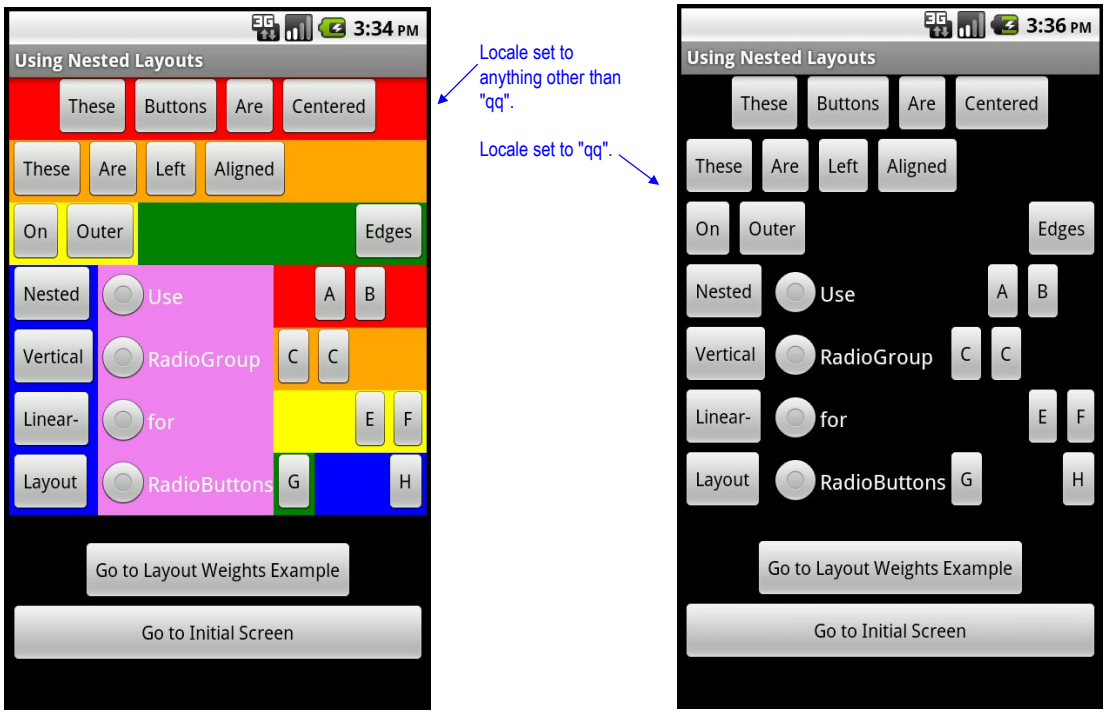
```
public class NestedLayoutsActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.nested_layouts);
    }

    ...

    // Event handlers for bottom two Buttons
}
```

There are two buttons on initial screen that invoke this same Activity. But, one sets the Locale to "qq" first.

Results



Layout Weight

Using android:layout_weight

- **Idea**
  - Assign numbers for android:layout_weight. Sizes given are proportional to those values.
- **Steps (for heights)**
  - Assign android:layout_height to 0dp
  - Use relative values for android:layout_weight
    - For example, if you have three nested entries with android:layout_weights of 1, 1, and 2, then they take up 25%, 25%, and 50% of the height of the parent.
  - Analogous approach to set widths
- **Common strategy**
  - Make the layout weights add up to 100, then treat them as percents. So, use 25, 25, and 50 instead of 1, 1, and 2 in the previous example. (Same effect, but clearer.)

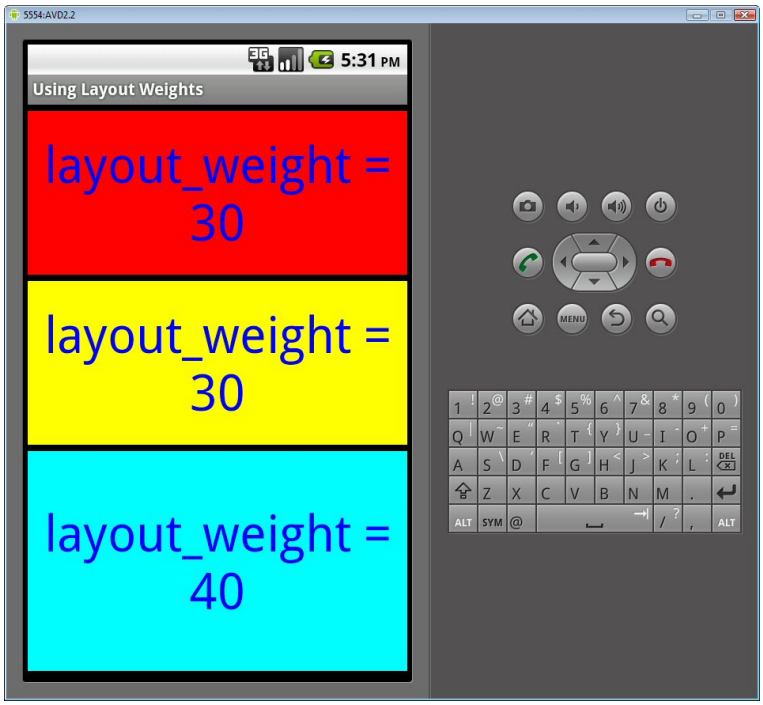
Layout File (res/layouts/layout_weights.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="…"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView android:layout_width="match_parent"
              android:layout_height="0dp"
              android:layout_weight="30"
              .../>
    <TextView android:layout_width="match_parent"
              android:layout_height="0dp"
              android:layout_weight="30"
              .../>
    <TextView android:layout_width="match_parent"
              android:layout_height="0dp"
              android:layout_weight="40"
              .../>
</LinearLayout>
```

Java Code

```
public class LayoutWeightsActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.layout_weights);
    }
}
```


Results



RelativeLayout

RelativeLayout

- **Idea**
  - Give ids to 1 or more key components (id="@+id/blah")
  - Position other components relative to those components
- **Most important XML attributes**
  - Aligning with container
    - android:layout_alignParentBottom (and Top, Right, Left)
    - android:layout_centerInParent (and centerHorizontal, centerVertical)
      - These all take "true" or "false" as values
  - Aligning with other component
    - android:layout_alignBottom (and Top, Right, Left)
    - android:layout_toLeftOf (and toRightOf), android:layout_above (and below)
      - These all take existing ids as values
        - » android:layout_align*Blah*="@id/existing_id" (@id, not @+id)

Referring to Existing IDs

- **First component**
  - <Button id="@+id/button_1"
    android:layout_alignParentRight="true" …/>

@+id for assigning a new id

- **Second component**
  - <Button android:layout_toLeftOf="@id/button_1" …/>

@id (no +) for referring to an existing id

- **Result**

Button 2 Button 1

Example Summary

- **General Approach**

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

<!-- Widgets and nested layouts -->

</RelativeLayout>
```

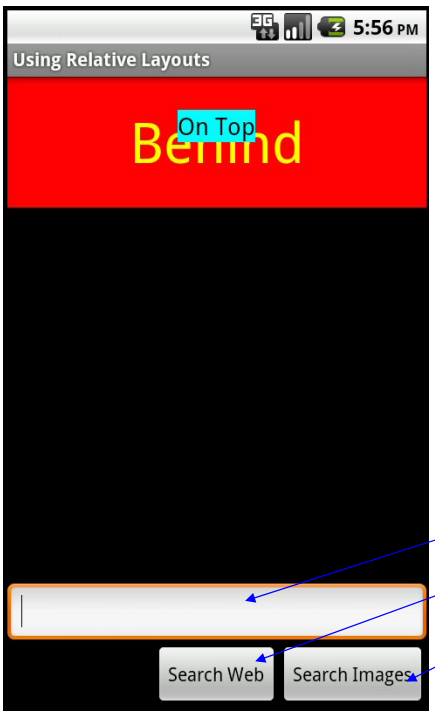
Example Details

TextView with width of match_parent and specific height. Goes at top since I didn't say otherwise. Has id.

TextView with android:layout_alignTop referring to first component. Moved down via android:layout_marginTop

```
<TextView android:id="@+id/behind"
        android:layout_width="match_parent"
        android:layout_height="100dp"
        android:background="#ff0000"
        android:textColor="#ffff00"
        android:textSize="42dp"
        android:text="Behind"
        android:gravity="center"/>
<TextView android:layout_alignTop="@id/behind"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#00ffff"
        android:textColor="#000000"
        android:textSize="18dp"
        android:text="On Top"
        android:layout_marginTop="25dp"
        android:layout_centerHorizontal="true"/>
```


Example Details




```
<Button android:id="@+id/image_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Search Images"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"/>
<Button android:layout_alignBottom="@id/image_button"
        android:layout_toLeftOf="@id/image_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Search Web"/>
<EditText android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_above="@id/image_button"
        android:layout_alignRight="@id/image_button"/>
```

EditText with android:layout_above referring to image button. Has width of match_parent.

Button with android:layout_alignBottom and android:layout_toLeftOf referring to image button. Has width of wrap_content.

Button with android:layout_alignParentBottom="true" and android:layout_alignParentRight="true". Has an id. Has width of wrap_content. This is the first Button defined.

TableLayout

TableLayout

- **Idea**
  - Put widgets or nested layouts in a grid. No borders.
  - Like HTML tables, the number of rows and columns is determined automatically, not explicitly specified.
  - Components are usually placed inside TableRow
- **Most important XML attributes (TableLayout)**
  - android:stretchColumns
    - An index or comma-separated list of indexes. Specifies the column or columns that should be stretched wider if the table is narrower than its parent. Indexes are 0-based.
  - android:shrinkColumns
    - Column(s) that should be shrunk if table is wider than parent.
  - android:collapseColumns
    - Column(s) to be totally left out. Can be programmatically put back in later.

TableRow

- **Idea**
  - Goes inside TableLayout to define a row.
    - Technically, elements between rows are permitted, but you can achieve same effect with a TableRow and android:layout_span.
- **Most important XML attributes of elements inside a TableRow**
  - android:layout_column
    - Normally, elements are placed in left-to-right order. However, you can use android:layout_column to specify an exact column, and thus leave earlier columns empty.
  - android:layout_span
    - The number of columns the element should straddle. Like colspan for HTML tables.
    - There is nothing equivalent to HTML's rowspan; you must use nested tables instead. See example.

Example Summary

- **General Approach**

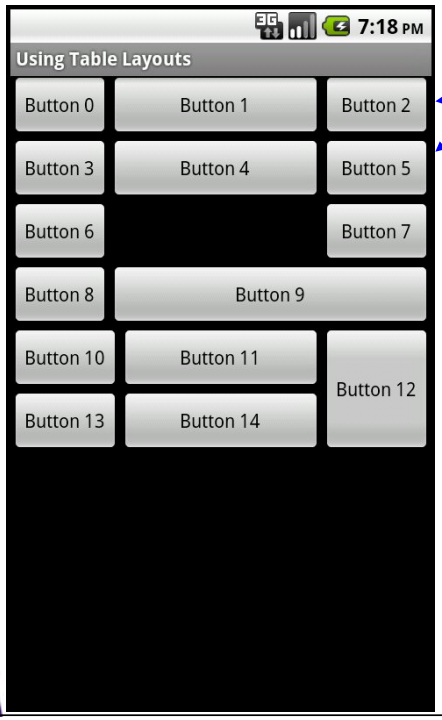
```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
    xmlns:android=
       "http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="1">

<TableRow>...</TableRow>
<TableRow>...</TableRow>
...
<TableRow>...</TableRow>

</TableLayout>
```

This is why the middle column is wider than the other two columns.

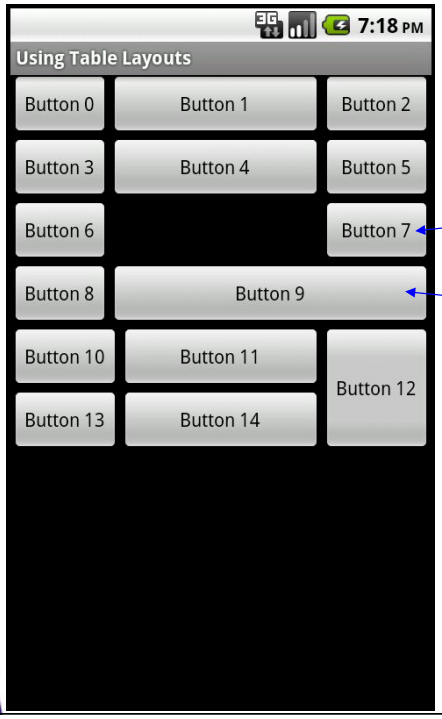
Example Details



Two TableRows, each with 3 Buttons. No special options.

```
<TableRow>
   <Button android:text="Button 0"/>
   <Button android:text="Button 1"/>
   <Button android:text="Button 2"/>
</TableRow>
<TableRow>
   <Button android:text="Button 3"/>
   <Button android:text="Button 4"/>
   <Button android:text="Button 5"/>
</TableRow>
```


Example Details

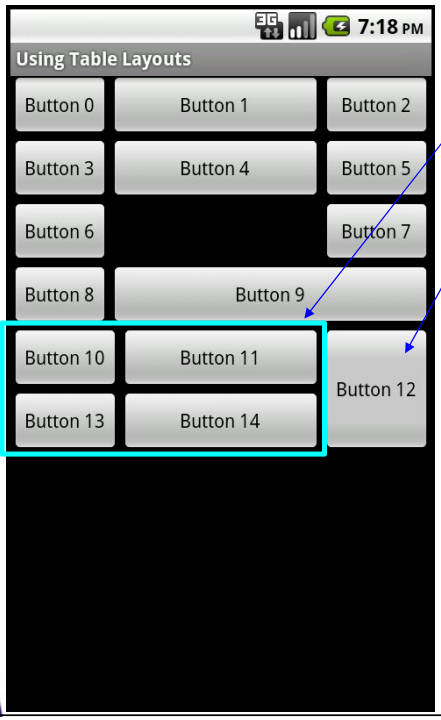


Button 7 uses android:layout_column="2". So, there is no entry at all for the middle column.

Button 9 uses android:layout_span="2".

```
<TableRow>
   <Button android:text="Button 6"/>
   <Button android:text="Button 7"
           android:layout_column="2"/>
</TableRow>
<TableRow>
   <Button android:text="Button 8"/>
   <Button android:text="Button 9"
           android:layout_span="2"/>
</TableRow>
```

Example Details



A nested table. Uses android:layout_span="2" so that it straddles two columns of the main table. Uses android:stretchColumns="1" so that the second column fills available space.

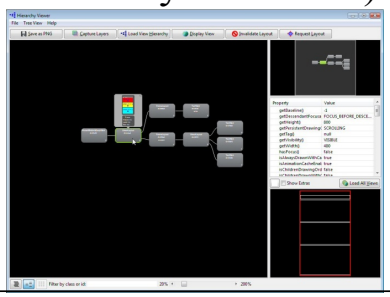
A Button. android:layout_height is match_parent so that it is the same height as table to its left. There is no option similar to HTML's colspan, so nested tables are needed to achieve this effect.

```
<TableRow>
  <TableLayout xmlns:android="..."
               android:layout_width="match_parent"
               android:layout_height="wrap_content"
               android:layout_span="2"
               android:stretchColumns="1">
    <TableRow>
      <Button android:text="Button 10"/>
      <Button android:text="Button 11"/>
    </TableRow>
    <TableRow>
      <Button android:text="Button 13"/>
      <Button android:text="Button 14"/>
    </TableRow>
  </TableLayout>
  <Button android:text="Button 12"
               android:layout_height="match_parent"/>
</TableRow>
```

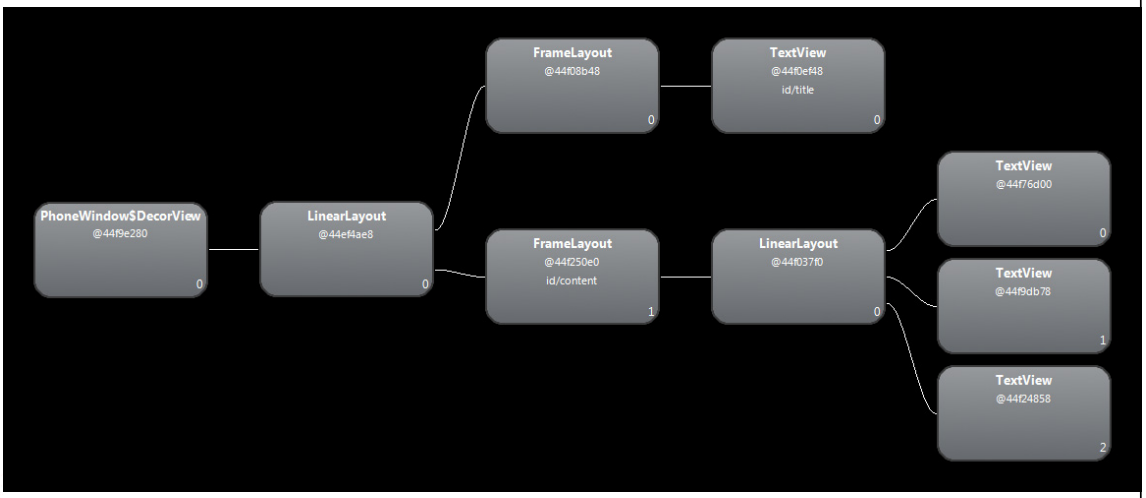

The Hierarchy Viewer

Hierarchy Viewer

- **Idea**
  - The Android distribution includes a program called hiearchyviewer that will show a graphical representation of Views and sub-Views. Useful for debugging and understanding nested layouts.
- **Details**
  - Start app in emulator. Go to screen of interest.
  - Go to *android-sdk*/tools (or, put this in your PATH)
  - Type hierarchyviewer
  - Click on Focused Window, then press Load View Hierarchy button
  - Explore!



Hierarchy View for RelativeLayout Example



Click on an entry to show which part of screen it corresponds to, and to get details about the XML attributes.

Details: http://developer.android.com/guide/developing/debugging/debugging-ui.html

Wrap-Up

Other Layouts

- **AbsoluteLayout**
  - From older versions; now deprecated; use RelativeLayout
- **FrameLayout**
  - For formatting a single item. Usually used explicitly with TabHost. Used internally by other layouts.
- **TabHost**
  - Combines tabs with switching Activities. Covered in later lecture on Intents and Activity switching.
- **ListView and GridView**
  - Not generalized layouts, but have somewhat similar role. Covered in later lecture.

More Reading

- **Tutorial: Declaring Layout**
  - http://developer.android.com/guide/topics/ui/declaring-layout.html
- **Tutorial: Hello Views**
  - http://developer.android.com/resources/tutorials/views/
    - Has sub-sections on LinearLayout, RelativeLayout, and TableLayout
- **Chapter: Working with Containers**
  - From *The Busy Coder's Guide to Android Development* by Mark Murphy.
    - http://commonsware.com/Android/
- **Chapter: User Interface Layout**
  - From *The Android Developer's Cookbook* by Steele & To

Summary

- **LinearLayout**
  - Ideas
    - One row or one column.
    - Nesting is key window-layout strategy
  - Key XML attributes
    - android:orientation, android:layout_weight
- **RelativeLayout**
  - Idea
    - Position later component relative to earlier one
  - Key XML attributes
    - android:layout_alignBottom (and similar), android:layout_toLeftOf (and similar)
- **TableLayout**
  - Idea
    - Put components in a grid
  - Key XML attributes for entries *inside* TableRow
    - android:layout_column, android:layout_span

Questions?