

# CS 193A

## Layout

# Sizing and Positioning

*How does the programmer specify where each component appears, how big each component should be, etc.?*

- **Absolute positioning** (C++, C#, others):
  - Programmer specifies exact pixel coordinates of every component.
  - "Put this button at (x=15, y=75) and make it 70x31 px in size."
- **Layout managers** (Java, Android):
  - Objects that decide where to position each component based on some general rules or criteria.
    - "Put these four buttons into a 2x2 grid and put these text boxes in a horizontal flow in the south part of the app."
  - More flexible and general; works better with a variety of devices.

# ViewGroup as layout

- ViewGroup superclass represents containers of widgets/views
  - layouts are described in **XML** and mirrored in Java code
  - Android provides several pre-existing layout managers; you can define your own **custom layouts** if needed
  - layouts can be **nested** to achieve combinations of features
- in the Java code and XML:
  - an **Activity** is a ViewGroup
  - various Layout classes are also ViewGroups
  - widgets can be added to a ViewGroup, which will then manage that widget's position/size behavior

# XML, in brief

- **XML** : a language for describing hierarchical text data. \*
  - Uses **tags** that consist of **elements** and **attributes**. Tags can be **nested**.
  - Some tags are opened and closed; others self-close.

`<element attr="value" attr="value"> ... </element>`  
`<element attr="value" attr="value" />` (self-closing)

- Example: \* XML is case-sensitive!

```
<!-- this is a comment -->
<course name="CS 193A" quarter="15wi">
  <instructor>Marty Stepp</instructor>
  <ta>none</ta>
</course>
```

# Changing layouts

- go to the **Text** view for your layout XML file
- modify the opening/closing tags to the new layout type, e.g. `LinearLayout`
- now go back to **Design** view and add widgets



The screenshot shows an IDE with two tabs: `MainActivity.java` and `activity_main.xml`. The `activity_main.xml` file is open, displaying XML code for a `LinearLayout`. The code is as follows:

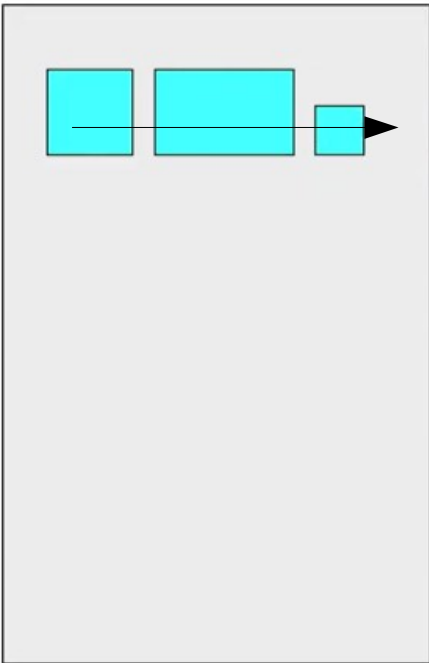
```
1 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2     xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
3     android:layout_height="match_parent" android:paddingLeft="16dp"
4     android:paddingRight="16dp"
5     android:paddingTop="16dp"
6     android:paddingBottom="16dp" tools:context=".MainActivity">
7
8 </LinearLayout>
```

The opening tag on line 1 is highlighted in blue, and the closing tag on line 8 is highlighted in yellow. A lightbulb icon is visible next to the closing tag, indicating a suggestion or warning.

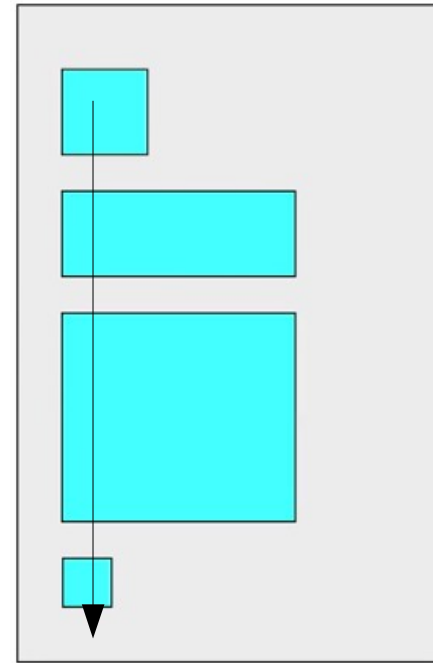
# LinearLayout ([link](#))

- lays out widgets/views in a single line
- **orientation** of horizontal (default) or vertical
- items do *not* wrap if they reach edge of screen!

horizontal



vertical



# LinearLayout example 1

```
<LinearLayout ...  
    android:orientation="horizontal"  
    tools:context=".MainActivity">  
    <Button ... android:text="Button 1" />  
    <Button ... android:text="Button 2 Hooray" />  
    <Button ... android:text="Button 3" />  
    <Button ... android:text="Button 4  
        Very Long Text" />  
</LinearLayout>
```



- In our examples, we'll use ... when omitting boilerplate code that is auto-generated by Android Studio and not relevant to the specific example at hand.

# LinearLayout example 2

```
<LinearLayout ...  
    android:orientation="vertical"  
    tools:context=".MainActivity">  
    <Button ... android:text="Button 1" />  
    <Button ... android:text="Button 2  
                                Hooray" />  
    <Button ... android:text="Button 3" />  
    <Button ... android:text="Button 4  
                                Very Long Text" />  
</LinearLayout>
```

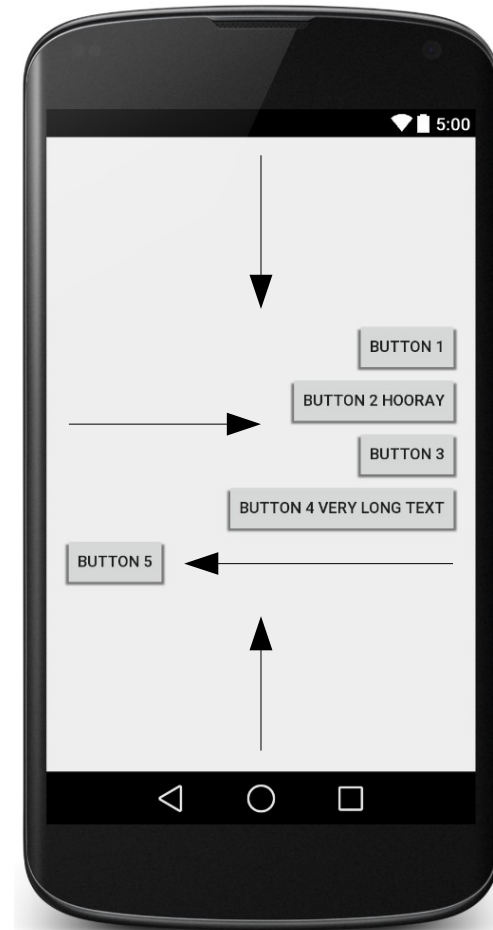




# Gravity

- **gravity**: alignment direction that widgets are pulled
  - top, bottom, left, right, center
  - combine multiple with |
  - set `gravity` on the layout to adjust all widgets; set `layout_gravity` on an individual widget

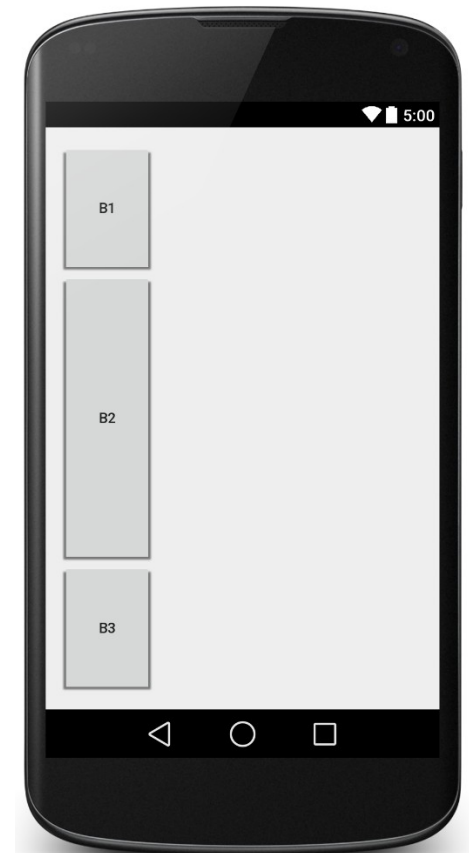
```
<LinearLayout ...  
    android:orientation="vertical"  
    android:gravity="center|right">  
    <Button ... android:text="Button 1" />  
    <Button ... android:text="Button 2 Hooray" />  
    <Button ... android:text="Button 3" />  
    <Button ... android:text="Button 4 Very Long Text" />  
    <Button ... android:text="Button 5"  
        android:layout_gravity="left" />  
</LinearLayout>
```



# Weight

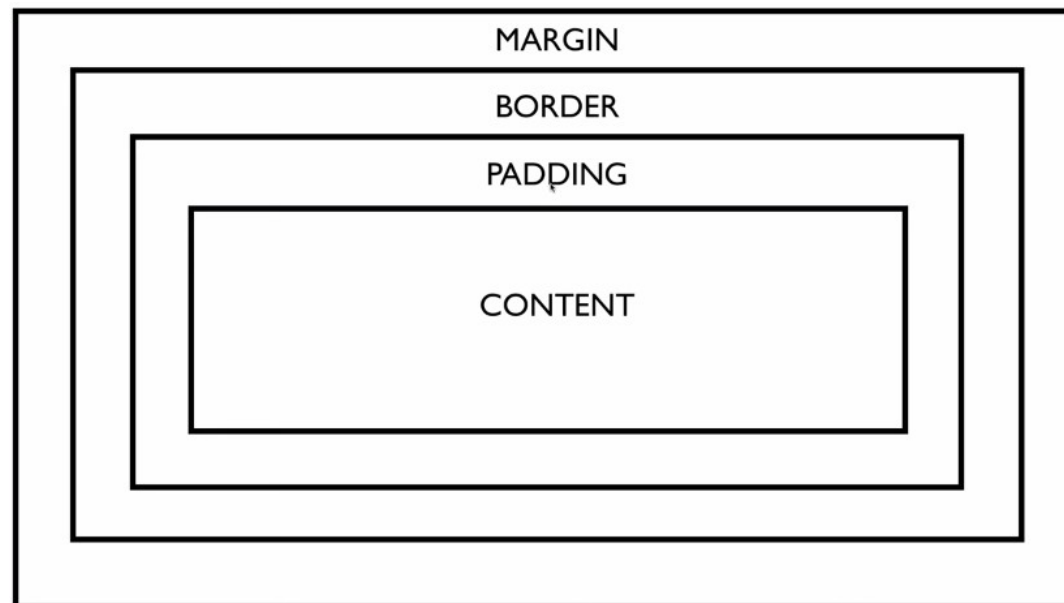
- **weight**: gives elements relative sizes by integers
  - widget with weight  $K$  gets  $K/\text{total}$  fraction of total size
  - cooking analogy: "2 parts flour, 1 part water, ..."

```
<LinearLayout ...  
    android:orientation="vertical">  
    <Button ... android:text="B1"  
                android:layout_weight="1" />  
    <Button ... android:text="B2"  
                android:layout_weight="3" />  
    <Button ... android:text="B3"  
                android:layout_weight="1" />  
</LinearLayout>
```



# Widget box model

- **content:** every widget or view has a certain size (width x height) for its content, the widget itself
- **padding:** you can artificially increase the widget's size by applying padding in the widget just outside its content
- **border:** outside the padding, a line around edge of widget
- **margin:** separation from neighboring widgets on screen



# Sizing an individual widget

- **width** and **height** of a widget can be:
  - `wrap_content` : exactly large enough to fit the widget's content
  - `match_parent` : as wide or tall as 100% of the screen or layout
  - a specific fixed width such as 64dp (*not usually recommended*)
    - *dp = device pixels; dip = device-independent pixels; sp = scaling pixels*

<Button ...

android:layout\_width="**match\_parent**"

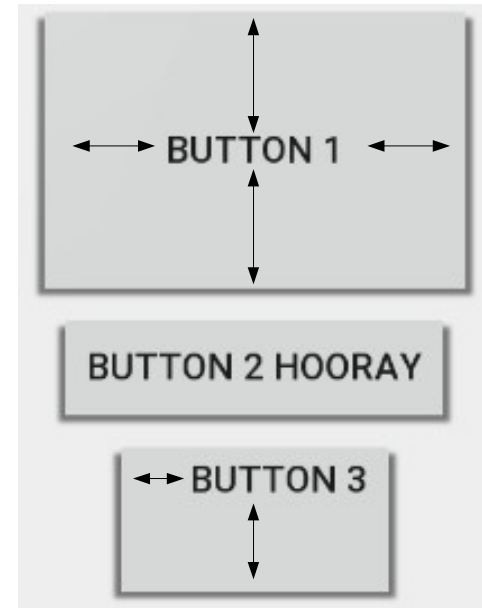
android:layout\_height="**wrap\_content**" />



# Padding

- **padding**: extra space *inside* widget
  - set padding to adjust all sides; paddingTop, Bottom, Left, Right for one side
  - usually set to specific values like 10dp  
*(some widgets have a default value ~16dp)*

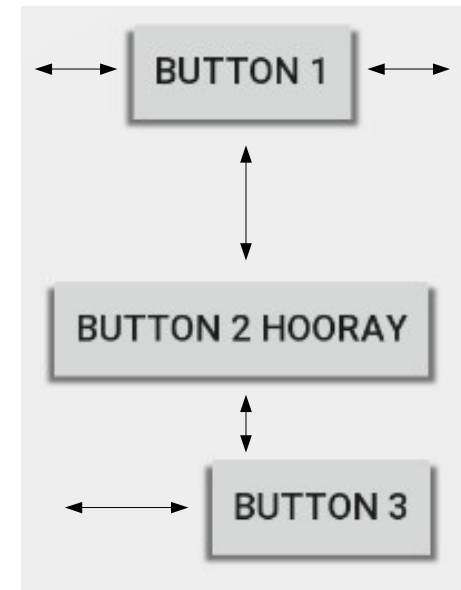
```
<LinearLayout ...  
    android:orientation="vertical">  
    <Button ... android:text="Button 1"  
        android:padding="50dp" />  
    <Button ... android:text="Button 2 Hooray" />  
    <Button ... android:text="Button 3"  
        android:paddingLeft="30dp"  
        android:paddingBottom="40dp" />  
</LinearLayout>
```



# Margin

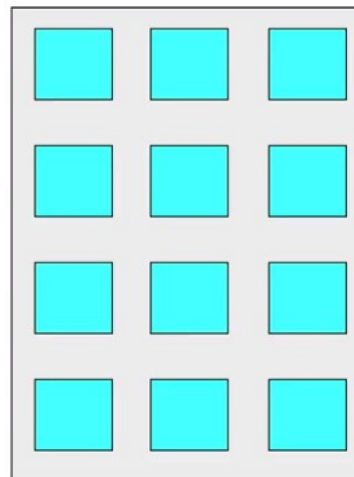
- **margin:** extra space *outside* widget to separate it from others
  - set `layout_margin` to adjust all sides;  
`layout_marginTop`, `Bottom`, `Left`, `Right`
  - usually set to specific values like `10dp`  
(*set defaults in `res/values/dimens.xml`*)

```
<LinearLayout ...  
    android:orientation="vertical">  
    <Button ... android:text="Button 1"  
        android:layout_margin="50dp" />  
    <Button ... android:text="Button 2 Hooray" />  
    <Button ... android:text="Button 3"  
        android:layout_marginLeft="30dp"  
        android:layout_marginTop="40dp" />  
</LinearLayout>
```



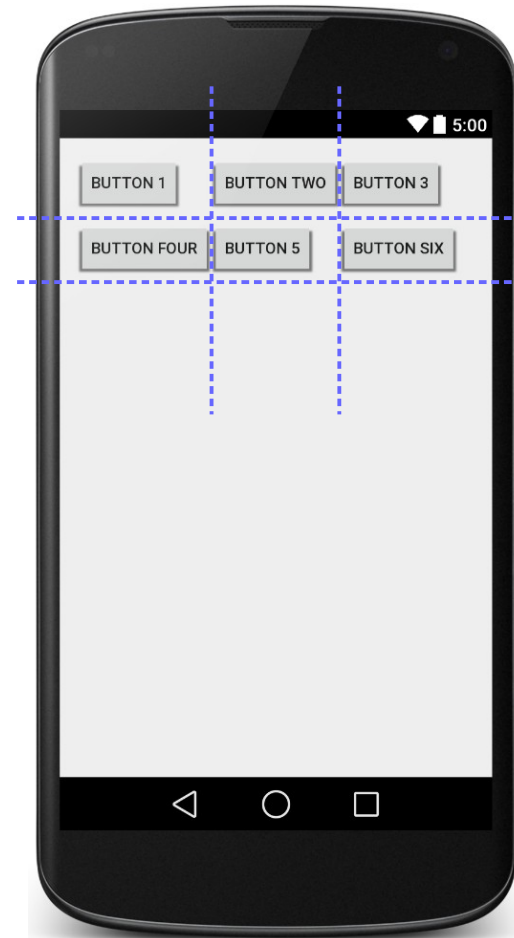
# GridLayout

- lays out widgets/views in lines of **rows** and **columns**
  - orientation attribute defines row-major or column-major order
  - introduced in Android 4; replaces older TableLayout
- by default, rows and columns are equal in size
  - each widget is placed into "next" available row/column index unless it is given an explicit `layout_row` and `layout_column` attribute
  - grid of 4 rows, 3 columns:



# GridLayout example 1

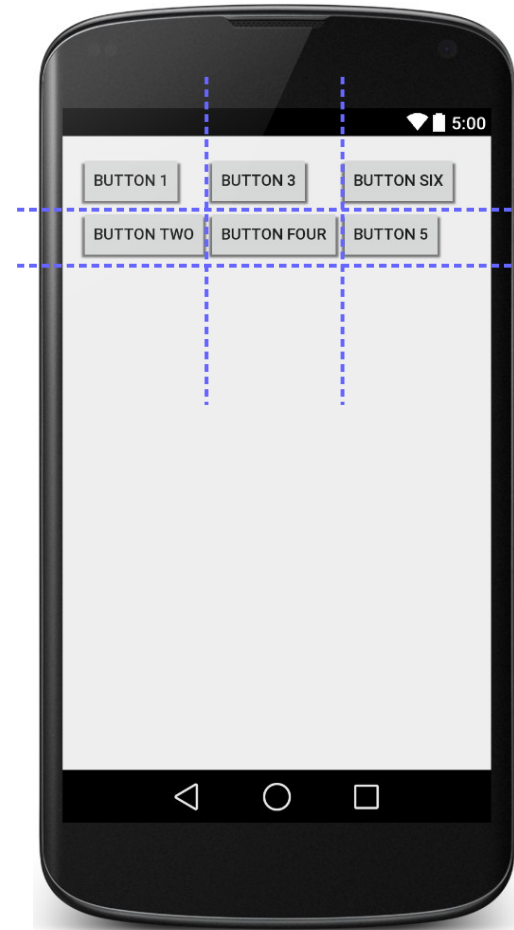
```
<GridLayout ...  
    android:rowCount="2"  
    android:columnCount="3"  
    tools:context=".MainActivity">  
    <Button ... android:text="Button 1" />  
    <Button ... android:text="Button Two" />  
    <Button ... android:text="Button 3" />  
    <Button ... android:text="Button Four" />  
    <Button ... android:text="Button 5" />  
    <Button ... android:text="Button Six" />  
</GridLayout>
```





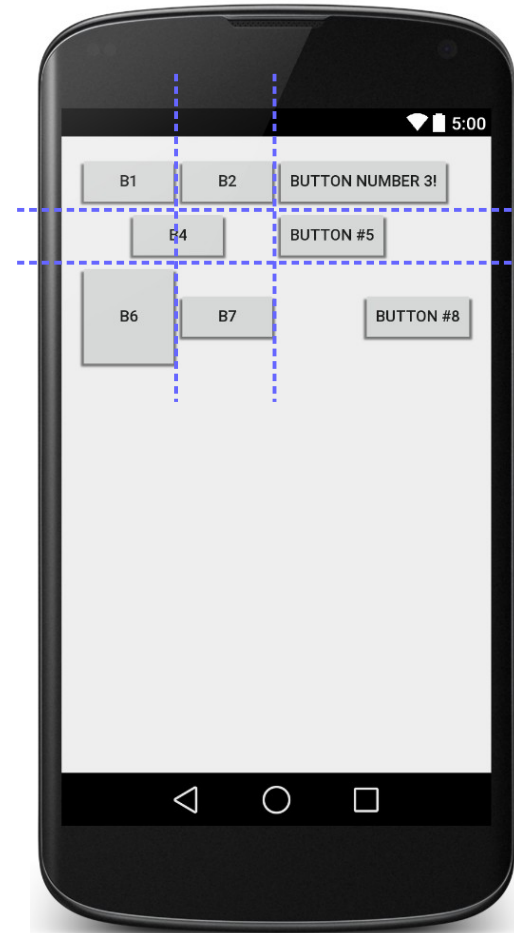
# GridLayout example 2

```
<GridLayout ...  
    android:rowCount="2"  
    android:columnCount="3"  
    android:orientation="vertical">  
    <Button ... android:text="Button 1" />  
    <Button ... android:text="Button Two" />  
    <Button ... android:text="Button 3" />  
    <Button ... android:text="Button Four" />  
    <Button ... android:text="Button 5"  
        android:layout_row="1"  
        android:layout_column="2" />  
    <Button ... android:text="Button Six"  
        android:layout_row="0"  
        android:layout_column="2" />  
</RelativeLayout>
```



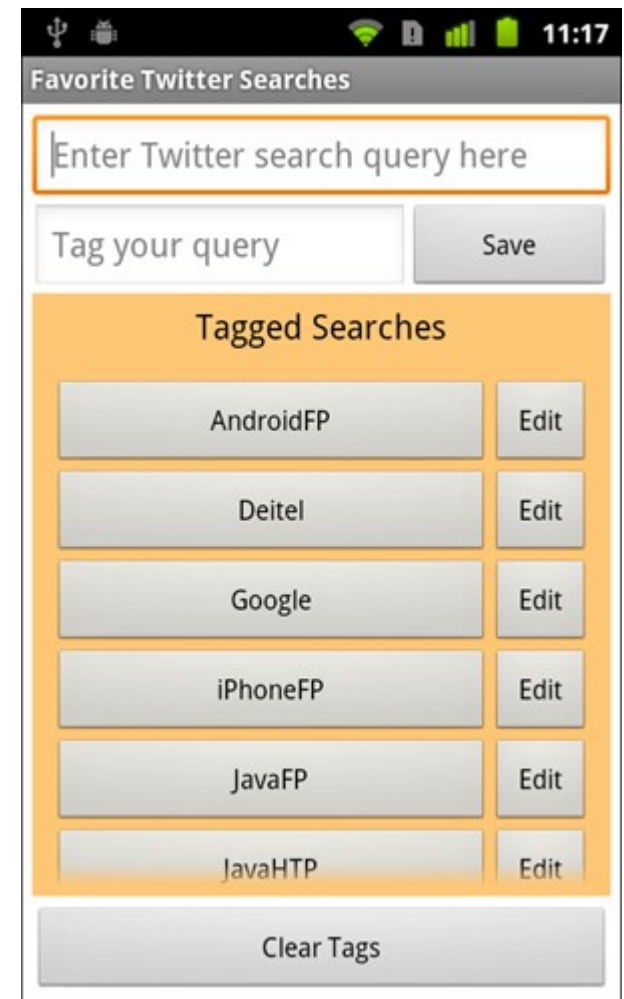
# GridLayout example 3

```
<GridLayout ...  
    android:rowCount="2"  
    android:columnCount="3">  
<Button ... android:text="B1" />  
<Button ... android:text="B2" />  
<Button ... android:text="Button Number 3!" />  
<Button ... android:text="B4"  
    android:layout_columnSpan="2"  
    android:layout_gravity="center" />  
<Button ... android:text="B5" />  
<Button ... android:text="B6"  
    android:layout_paddingTop="40dp"  
    android:layout_paddingBottom="40dp" />  
<Button ... android:text="B7" />  
<Button ... android:text="Button #8"  
    android:layout_gravity="right" />  
</RelativeLayout>
```



# Nested layout

- to produce more complicated appearance, use a **nested** layout
  - (layouts inside layouts)
- what layout(s) are used to create the appearance at right?
  - overall activity: \_\_\_\_\_
  - internal layouts: \_\_\_\_\_



# Nested layout template

```
<OuterLayoutType ...>
```

```
    <InnerLayoutType ...>
```

```
        <Widget ... />
```

```
        <Widget ... />
```

```
    </InnerLayoutType>
```

```
    <InnerLayoutType ...>
```

```
        <Widget ... />
```

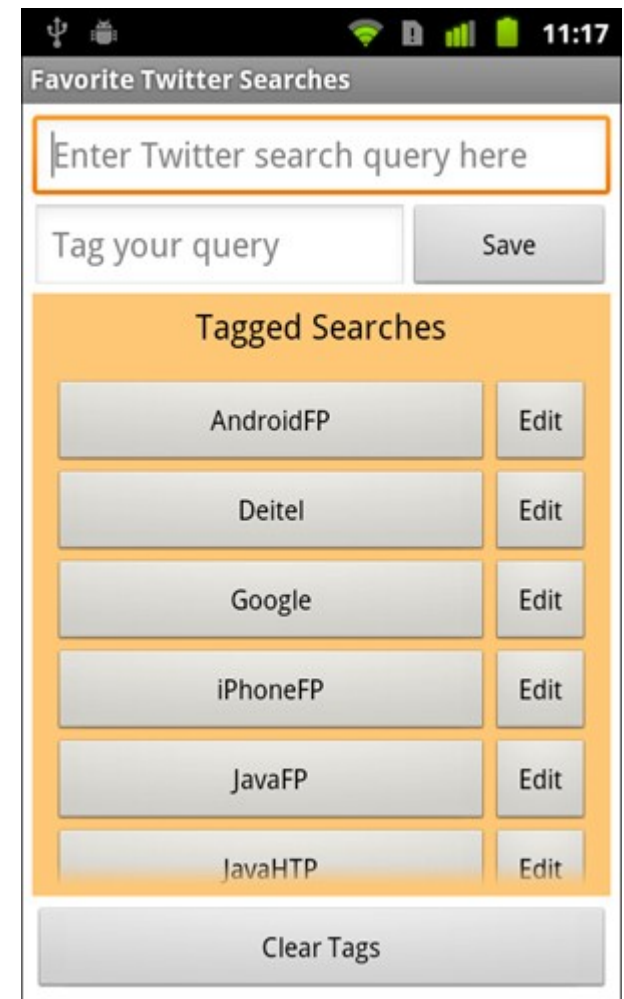
```
        <Widget ... />
```

```
    </InnerLayoutType>
```

```
    <Widget ... />
```

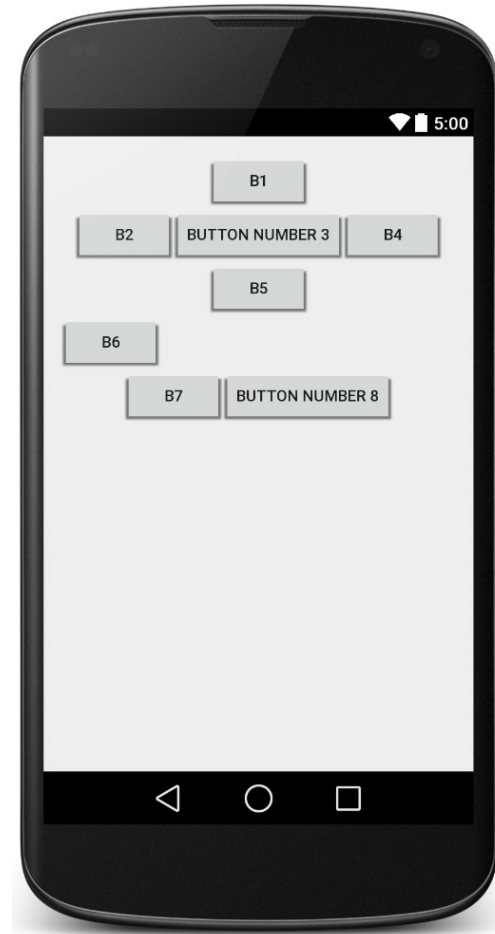
```
    <Widget ... />
```

```
</OuterLayoutType>
```



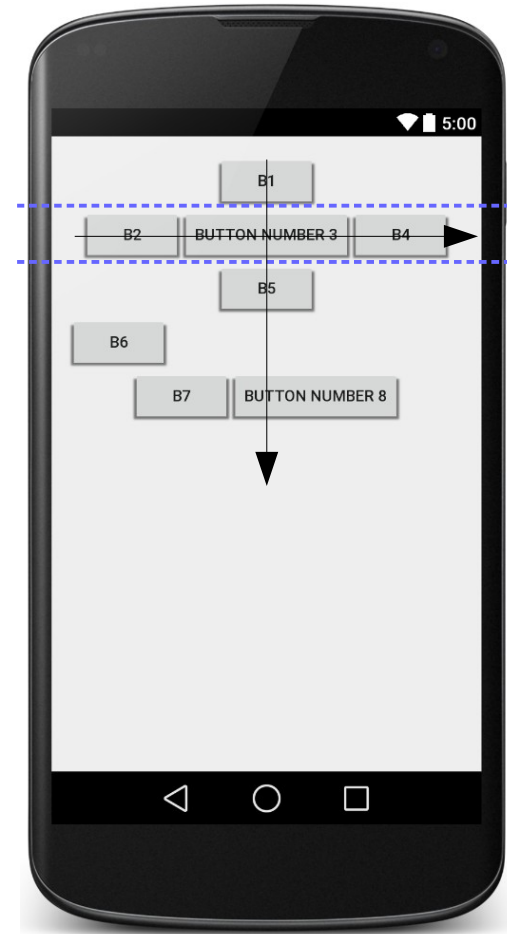
# Nested layout exercise

- Write the layout XML necessary to create the following app UI.
  - How many overall layouts are needed?
  - Which widgets go into which layouts?
  - ...



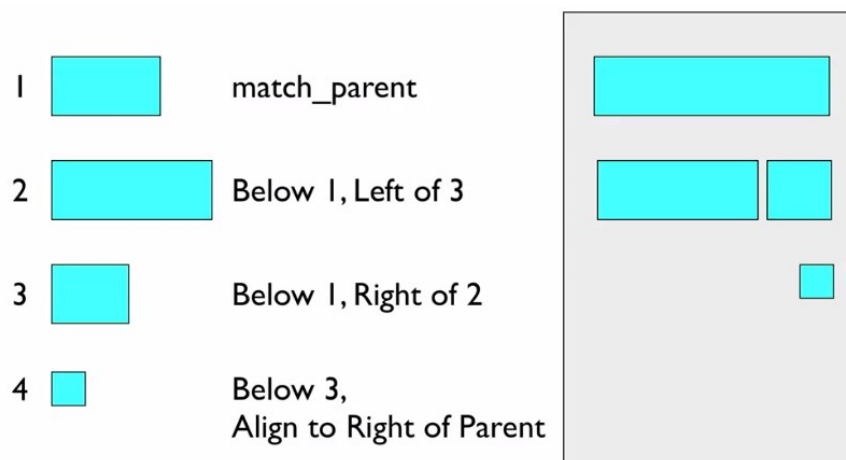
# Nested layout solution

```
<LinearLayout ...  
    android:orientation="vertical" android:gravity="center|top">  
    <Button ... android:text="B1" />  
    <LinearLayout ...  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:orientation="horizontal"  
        android:gravity="center|top">  
        <Button ... android:text="B2" />  
        <Button ... android:text="Button Number 3" />  
        <Button ... android:text="B4" />  
    </LinearLayout>  
    <Button ... android:text="B5" />  
    <Button ... android:text="B6" android:layout_gravity="left" />  
    <LinearLayout ...  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:orientation="horizontal"  
        android:gravity="center|top">  
        <Button ... android:text="B7" />  
        <Button ... android:text="Button Number 8" />  
    </LinearLayout>  
</LinearLayout>
```



# RelativeLayout ([link](#))

- each widget's position and size are relative to other views
  - relative to "parent" (the activity itself)
  - relative to other widgets/views
  - x-positions of reference: left, right, center
  - y-positions of reference: top, bottom, center
- intended to reduce the need for nested layouts



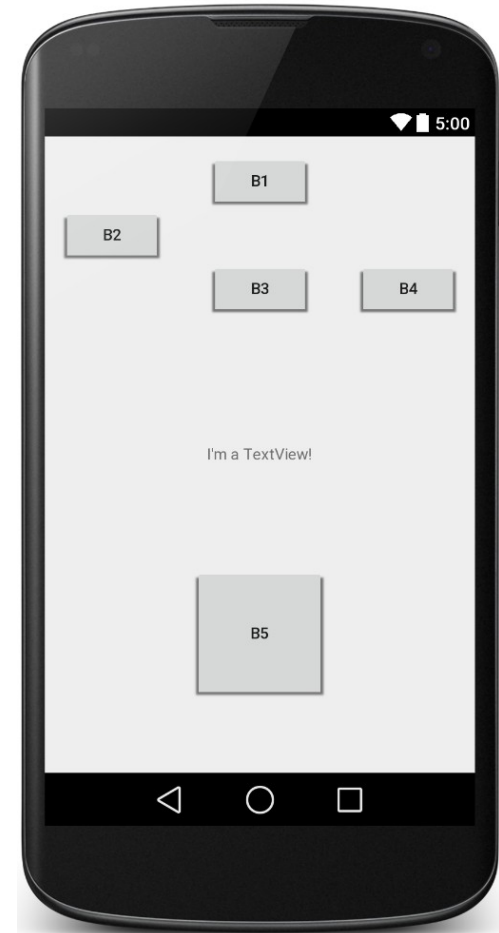
# Relative anchor points

- properties for x/y relative to **another widget**:
  - **layout\_below**, **above**, **toLeftOf**, **toRightOf**
    - set these to the ID of another widget in the format "**@id/*theID***"  
(*obviously, the given widget must have an ID for this to work*)
- properties for x/y relative to layout **container** (the activity):
  - **layout\_alignParentTop**, **Bottom**, **Left**, **Right**
    - set these flags to a boolean value of "true" to enable them
  - **layout\_centerHorizontal**, **Vertical**, **InParent**
    - set these flags to "true" to center the control within its parent in a dimension



# RelativeLayout example 1

```
<RelativeLayout ... >
    <Button ... android:id="@+id/b1" android:text="B1"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />
    <Button ... android:id="@+id/b2" android:text="B2"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/b1" />
    <Button ... android:id="@+id/b3" android:text="B3"
        android:layout_centerHorizontal="true"
        android:layout_below="@+id/b2" />
    <Button ... android:id="@+id/b4" android:text="B4"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/b2" />
    <TextView ... android:id="@+id/tv1"
        android:text="I'm a TextView!"
        android:layout_centerInParent="true" />
    <Button ... android:id="@+id/b5" android:text="B5"
        android:padding="50dp"
        android:layout_centerHorizontal="true"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="50dp" />
</RelativeLayout>
```



# FrameLayout ([link](#))

- meant to hold only a single widget inside, which occupies the entirety of the activity
  - most commonly used with layout fragments (seen later)
  - less useful for more complex layouts

*(can put in multiple items and move them to "front" in Z-order)*

```
<FrameLayout ... >  
    <ImageView  
        android:src="@drawable/jellybean"  
        ... />  
</FrameLayout>
```

