Android UI

Plan

- UI as a resource
- UI elements
  - Widgets
  - Layouts
    - Linear
    - Grid
    - Relative
    - Constrained
- The API

The application resources

- Main idea: keep all the elements of the visual presentation of the app **separated from the code**
- Example:
  - Graphical Interface
  - Text of each component (buttons, field texts etc)
  - Icons, images
  - Other images, video, pdf, webpages…
- Each element is added to the app with an unique integer ID
- The class `R.java` collects all the IDs

```
//Load the graphical layout activity_main.xml
setContentView( R.layout.activity_main );
```

Resources

R.java

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */
package com.example.helloworld;

public final class R {
    public static final class id {
        public static final int action_settings=0x7f080001;
        public static final int textbox=0x7f080000;
    }
    public static final class layout {
        public static final int activity_main=0x7f030000;
    }
    public static final class menu {
        public static final int main=0x7f070000;
    }
    public static final class string {
        public static final int action_settings=0x7f050001;
        public static final int app_name=0x7f050000;
        public static final int hello_world=0x7f050002;
    }
```

XML layout description

```
<RelativeLayout

    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent”
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/textbox"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world” >
    </TextView>

</RelativeLayout>
```

Unique ID of the element

Element content from `string.xml`

activity_main.xml

Other resources

- The file `values/strings.xml` collects a list of strings that can be used for the text of the GUI

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
</resources>
```

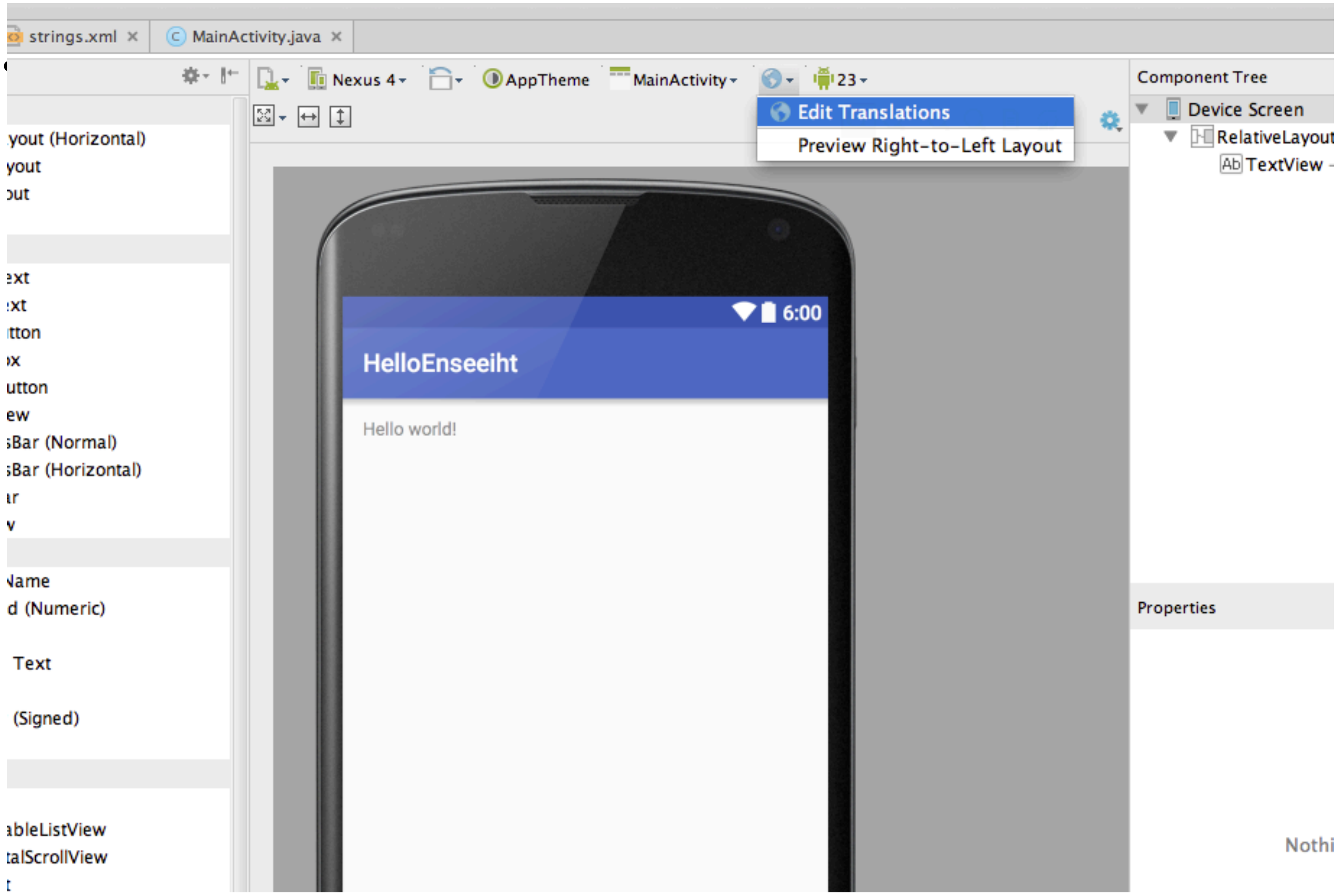
values/strings.xml

```xml
…
    <TextView
        android:id="@+id/textbox"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/hello_world” >
     </TextView>

</RelativeLayout>
```

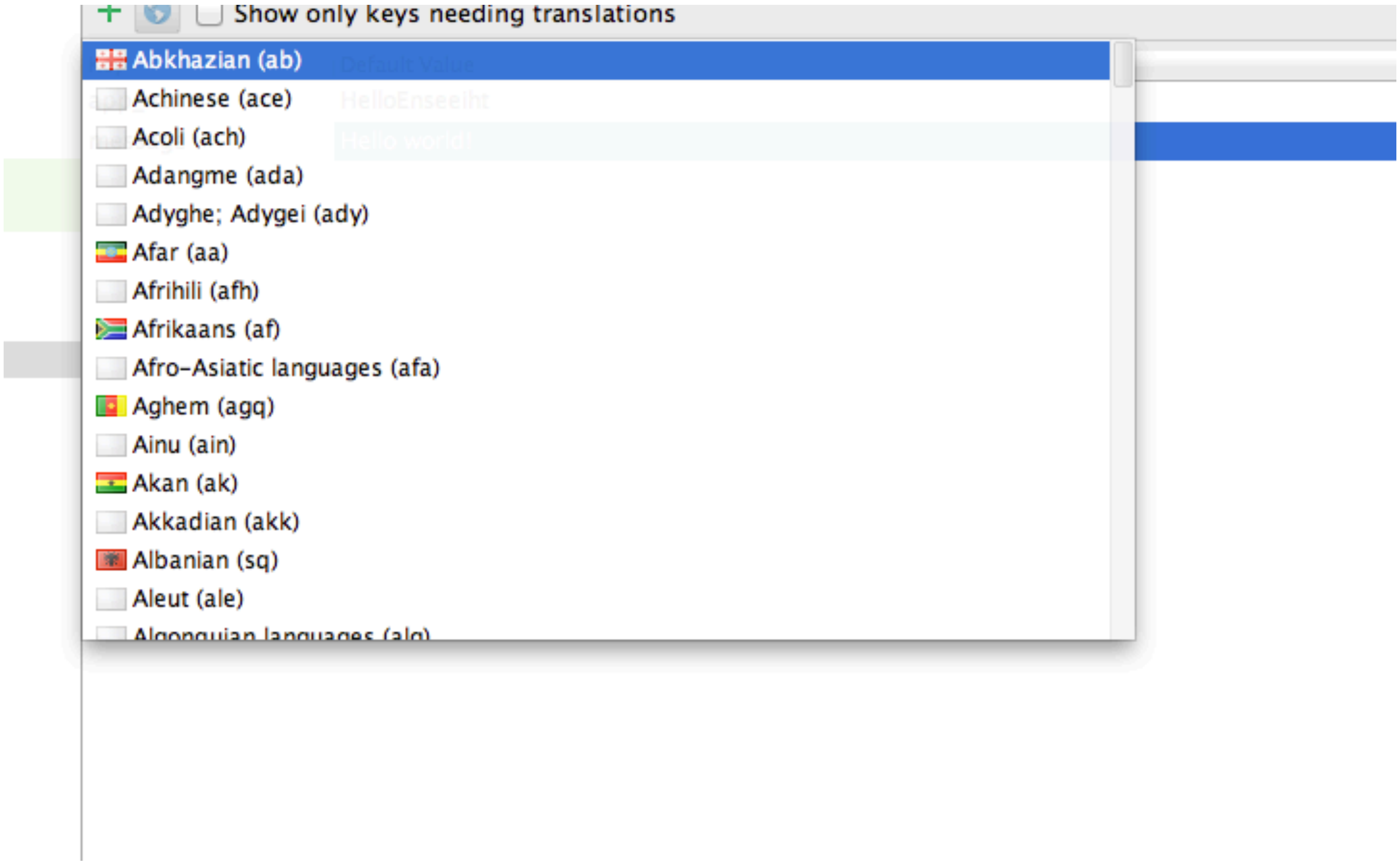
activity_main.xml

Why is this important?
Application Localisation!

Application Localisation - editor

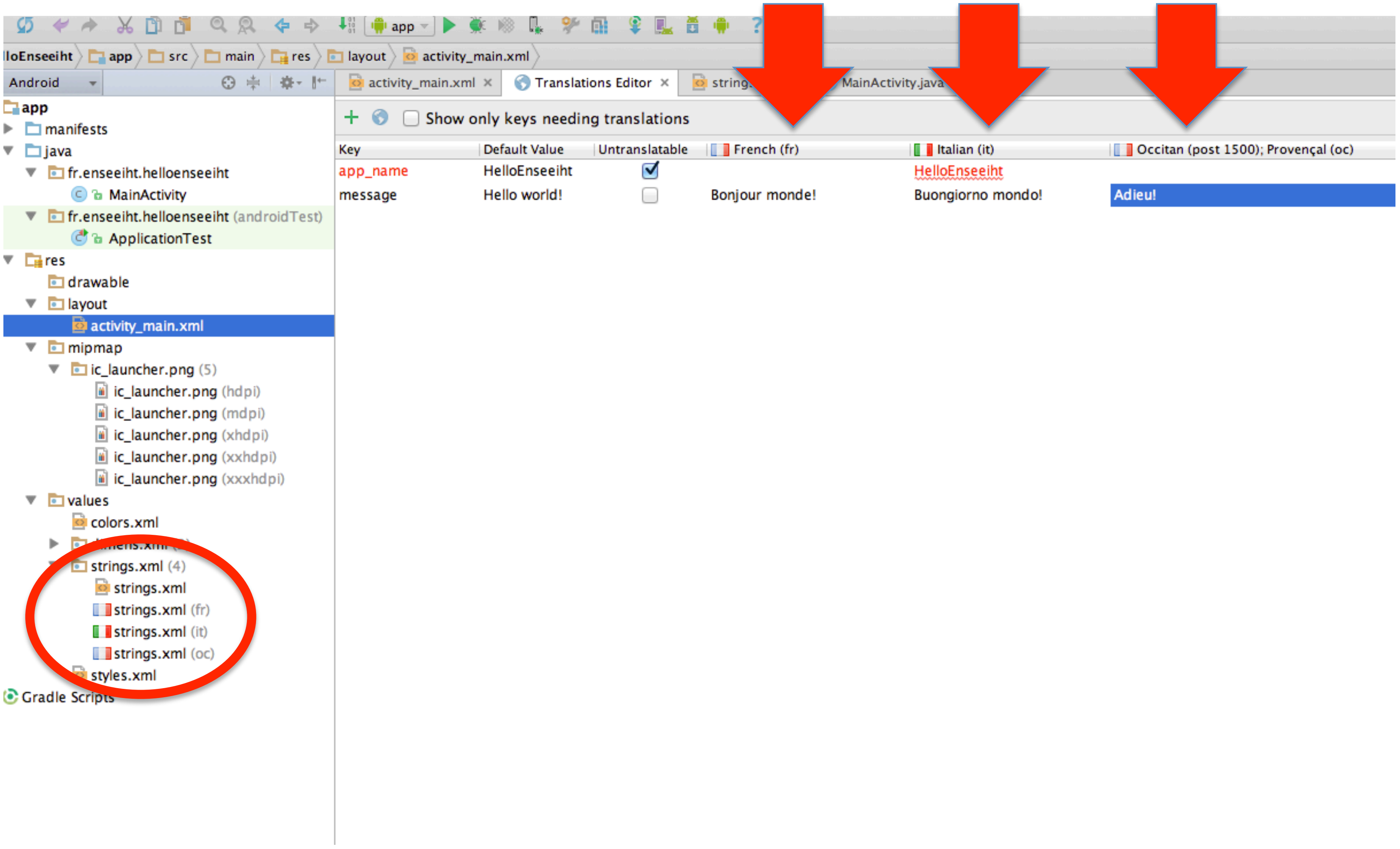


Translation editor



Abkhazian (ab)
Achinese (ace)
Acoli (ach)
Adangme (ada)
Adyghe; Adygei (ady)
Afar (aa)
Afrihili (afh)
Afrikaans (af)
Afro-Asiatic languages (afa)
Aghem (agq)
Ainu (ain)
Akan (ak)
Akkadian (akk)
Albanian (sq)
Aleut (ale)

Translation editor



Key	Default Value	Untranslatable	French (fr)	Italian (it)	Occitan (post 1500); Provençal (oc)
app_name	HelloEnseeiht	✓		HelloEnseeiht	
message	Hello world!	☐	Bonjour monde!	Buongiorno mondo!	Adieu!

Example of language localisation

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="message">Hello world!</string>
</resources>
```

values/strings.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">HelloWorld</string>
    <string name="message">Bonjour Monde!</string>
</resources>
```

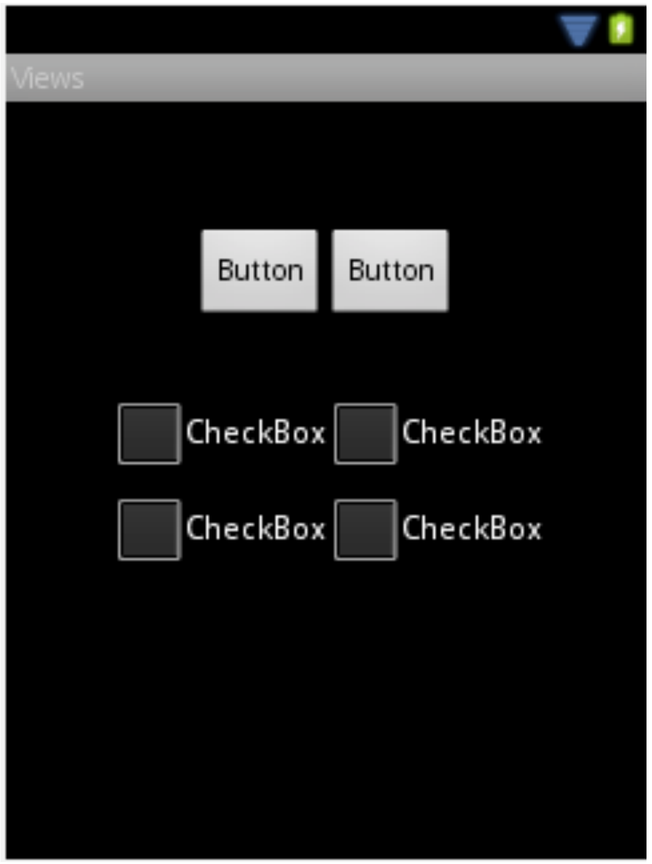
values-fr/strings.xml

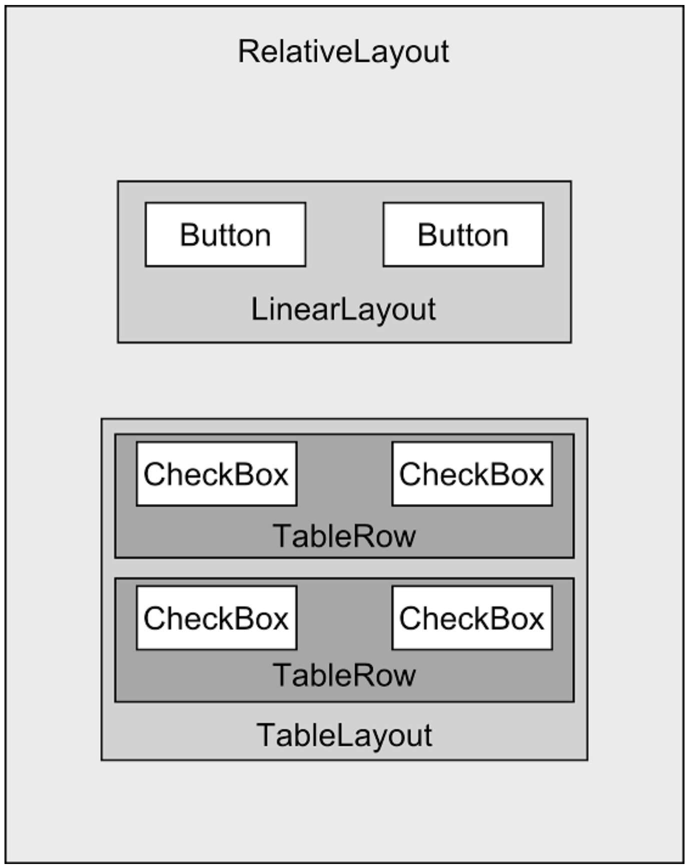
Whenever the user changes the global language of the device the relevant text will be displayed (if available… by default the one in `values`)

Plan

- UI as a resource
- UI elements
  - Widgets
  - Layouts
    - Linear
    - Grid
    - Relative
    - Constrained
- The API

The user interface





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.

Managing the UI in Android

- The most common way: **XML layout file**
  - an XML layout file saved in your application resources.
  - Keep the design of your user interface separated from the code that defines the activity's behavior.

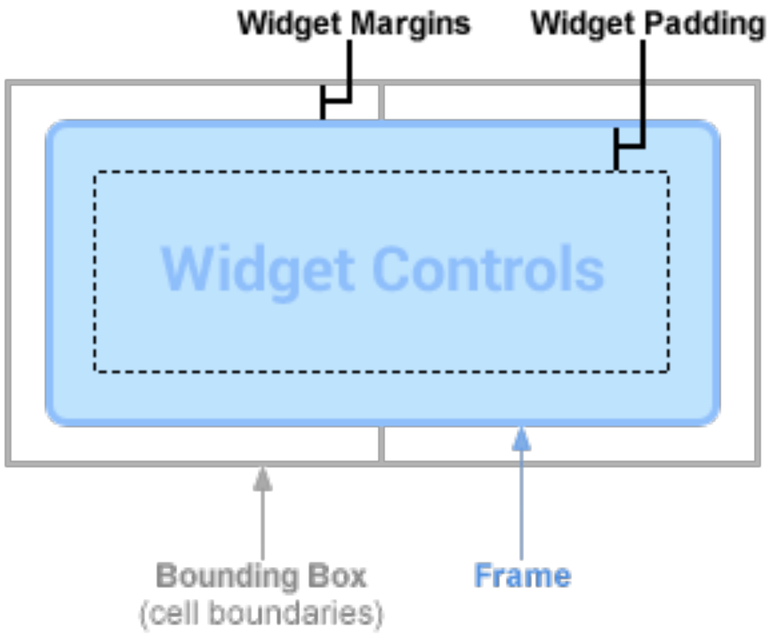
- The hard way: **programmatically**
  - Build the hierarchy inside the code creating the objects from the relevant classess
  - Add custom widget and layout subclassing the class view .
  - Good luck with that!

Plan

- UI as a resource
- UI elements
  - Widgets
  - Layouts
    - Linear
    - Grid
    - Relative
    - Constrained
- The API

Widget box model

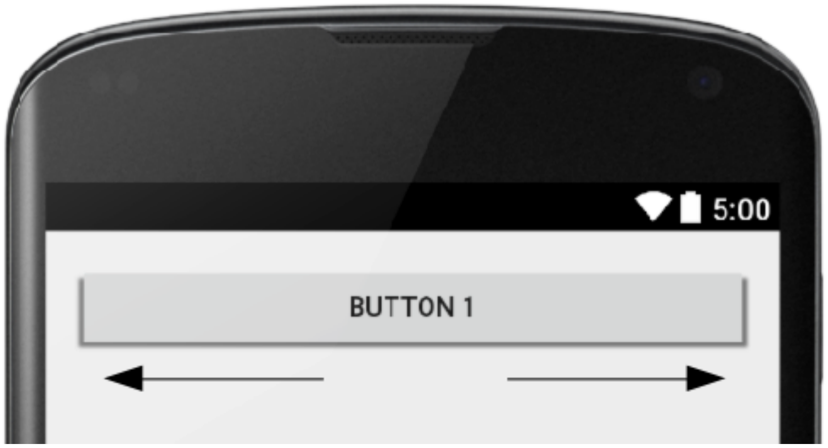
- **content**: every widget or view has a certain size (width x height) for its content, the widget itself
- **padding**: you can artificially increase the widget's size by applying padding in the widget just outside its content
- **frame**: outside the padding, a line around edge of widget
- **margin**: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



Widget box model

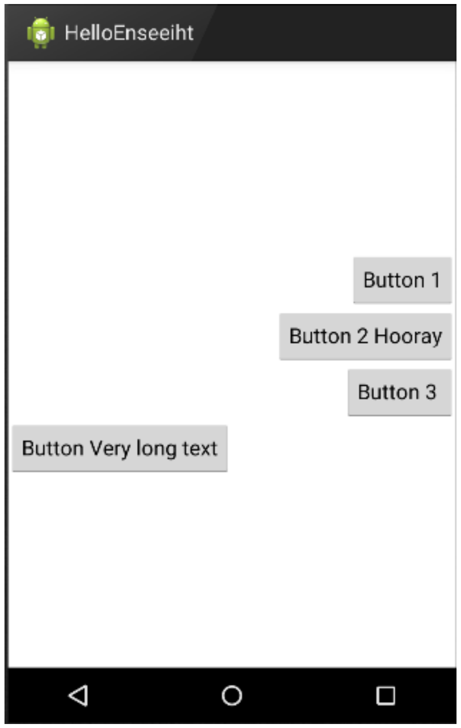
- `width` and `height` of a widget can be:
  - `wrap_content` : exactly large enough to fit the widget's content
  - `match_parent` : as wide or tall as 100% of the screen or layout
  - a specific fixed width such as 64dp *(not usually recommended)*
    dp = device pixels; dip = device-independent pixels;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

```
<Button ... android:text="Button 1"
           android:layout_width="match_parent"
           android:layout_height="wrap_content" />
```



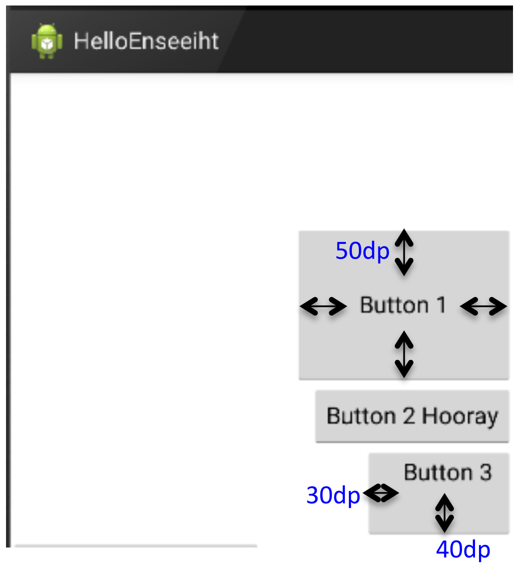
Widget box model

```
<LinearLayout ...
    android:orientation=”vertical” >

   <Button ... android:text="Button 1” />

   <Button ... android:text="Button 2 Hooray” />

   <Button ... android:text="Button 3" />
   <Button ... android:text="Button Very long text"
        android:layout_gravity="left"/>
```



Widget box model

- **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

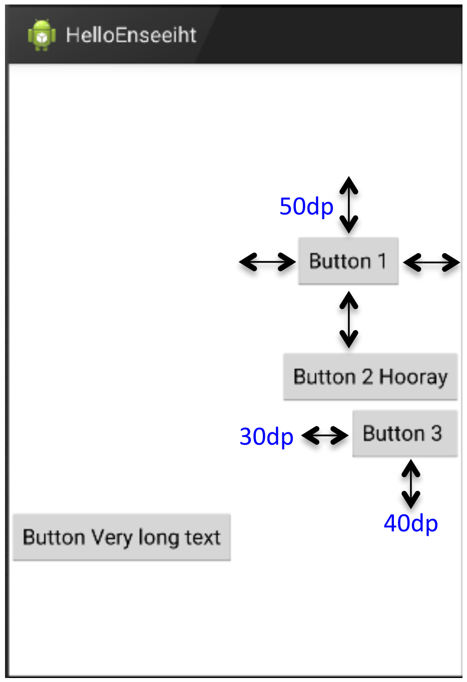


```
<LinearLayout ...
     android:orientation=”vertical” >

    <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" />
```

Widget box model

- **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

```
<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_marginBottom="40dp" />
```



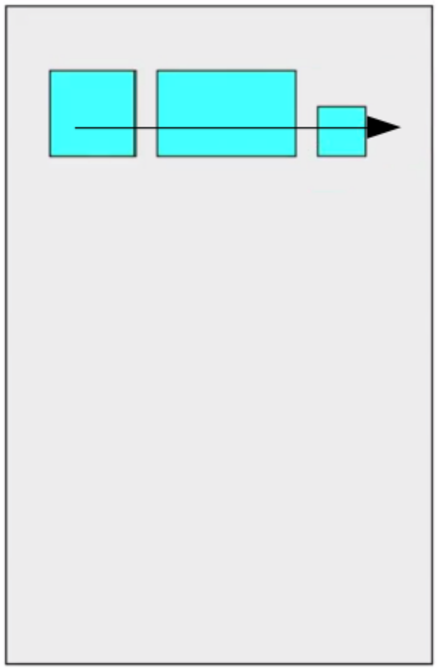
Plan

- UI as a resource
- UI elements
  - Widgets
  - Layouts
    - Linear
    - Grid
    - Relative
    - Constrained
- The API

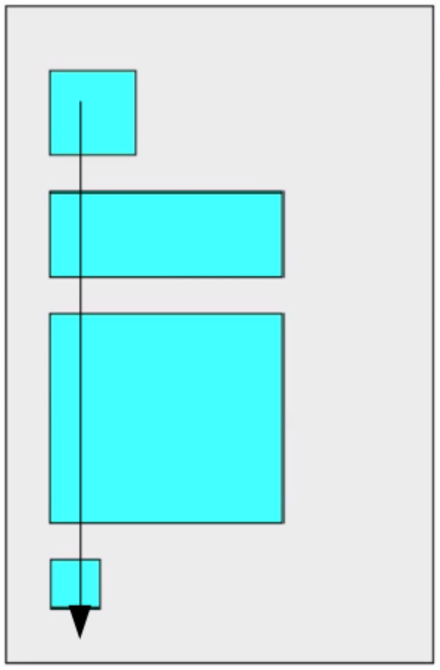
Linear Layout

- `LinearLayout`
- aligns all children in a single direction, vertically or horizontally.
- `android:orientation=”[vertical|horizontal]”`

Horizontal

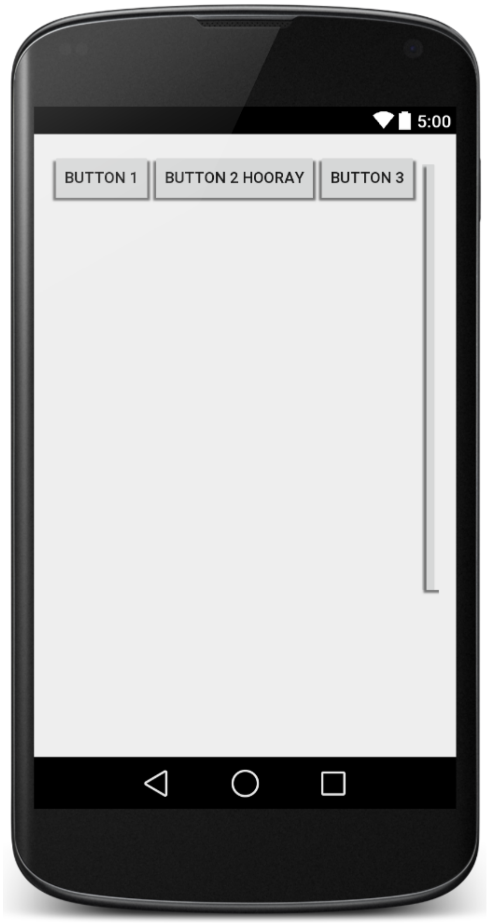


Vertical



Linear Layout - horizontal

```
<LinearLayout ...
    android:orientation="horizontal" >

   <Button ... android:text="Button 1" />

   <Button ... android:text="Button 2 Hooray" />

   <Button ... android:text="Button 3 " />

   <Button ... android:text="Button Very long text" />

</LinearLayout>
```

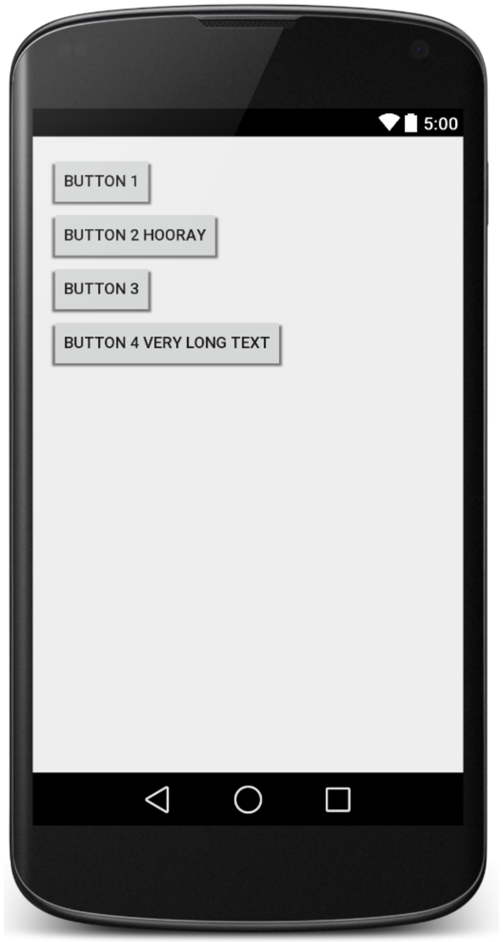


Just for the sake of the example the content of the button is written explicitly here.
USE STRINGS INSTEAD!!!

Linear Layout – vertical

```
<LinearLayout ...
    android:orientation="vertical" >

   <Button ... android:text="Button 1" />

   <Button ... android:text="Button 2 Hooray" />

   <Button ... android:text="Button 3 " />


   <Button ... android:text="Button Very long text" />

</LinearLayout>
```

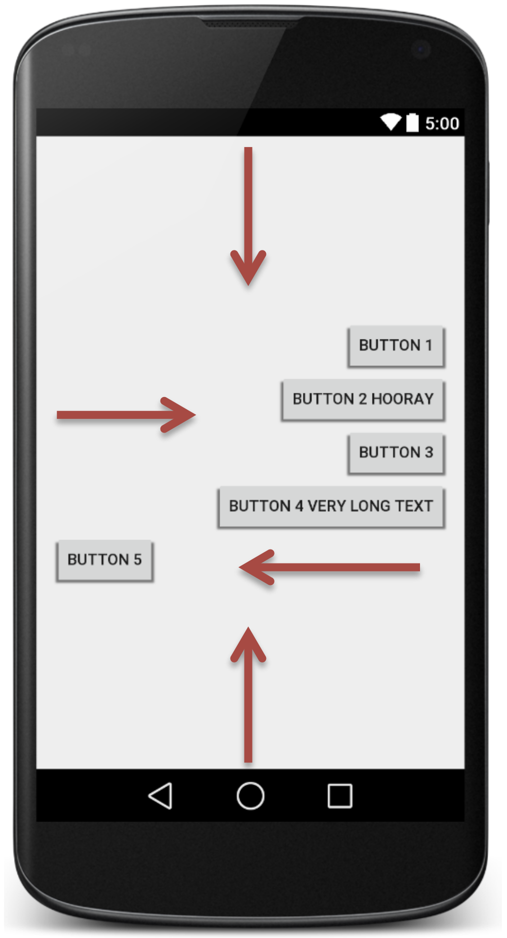
Just for the sake of the example the content of the button is written explicitly here.
USE STRINGS INSTEAD!!!



Gravity attribute

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

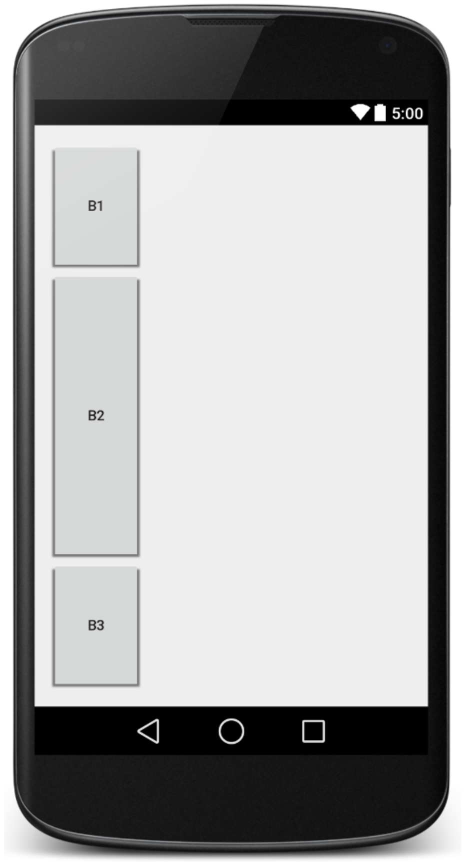
```
<LinearLayout ...
    android:orientation=”vertical"
    android:gravity=”center|right" >

   <Button ... android:text="Button 1" />

   <Button ... android:text="Button 2 Hooray" />

   <Button ... android:text="Button 3 " />

   <Button ... android:text="Button 5"
               android:layout_gravity=”left" />

</LinearLayout>
```



Weight attribute

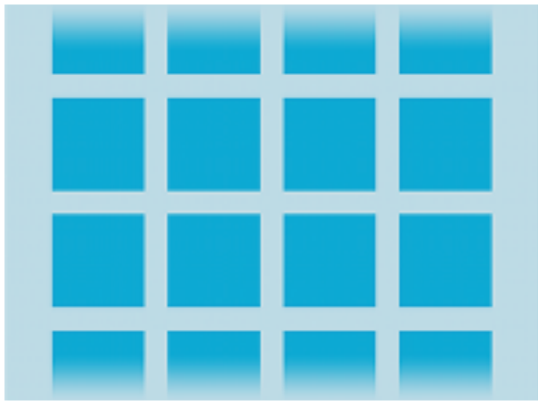
- **weight**: gives elements relative sizes by integers
  - widget with weight ***K*** gets ***K***/total fraction of total size

```
<LinearLayout ...
    android:orientation=”vertical” >

   <Button ... android:text="Button 1"
               android:layout_weight=”1" />

   <Button ... android:text="Button 2 Hooray"
               android:layout_weight=”3" />

   <Button ... android:text="Button 3 "
               android:layout_weight=”1" />


  </LinearLayout>
```



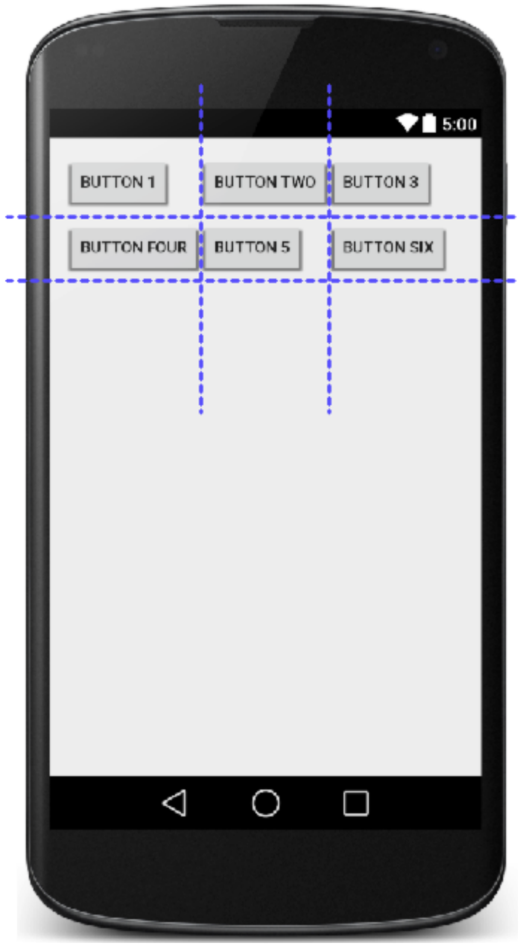
Grid Layout

- 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



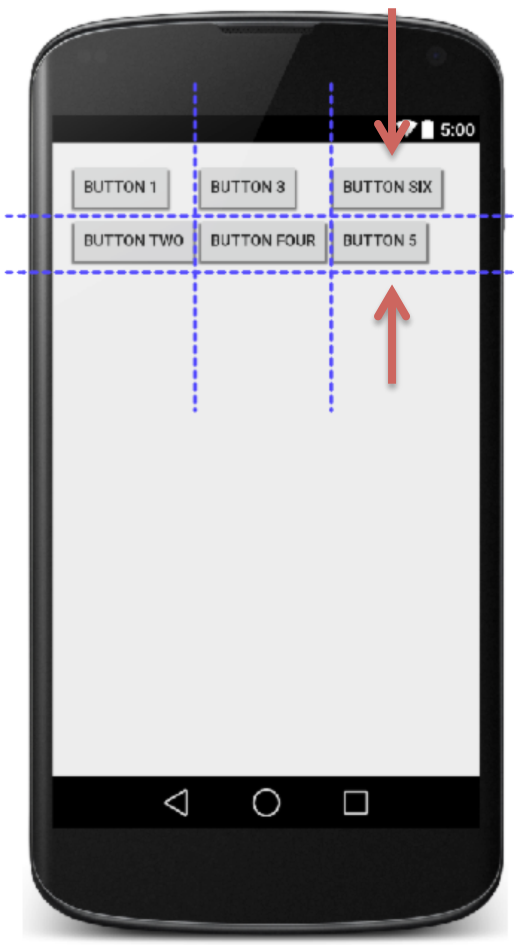
Example

```
<GridLayout ...
    android1:columnCount="3"
    android1:rowCount="2" >

    <Button ... android1:text="Button 1" />

    <Button ... android1:text="Button two" />

    <Button ... android1:text="Button 3" />

    <Button ... android1:text="Button four" />

    <Button ... android1:text="Button 5" />

    <Button ... android1:text="Button six" />

</GridLayout>
```



Example

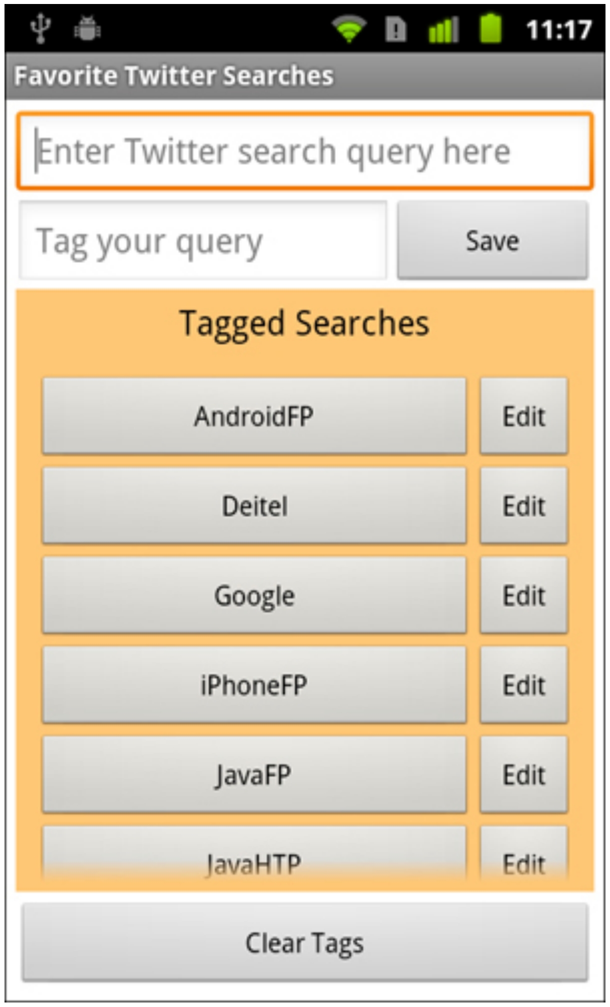
```
<GridLayout ...
    android:columnCount="3"
    android:rowCount="2"
    android:orientation=”vertical" >

    <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"/>

</GridLayout>
```



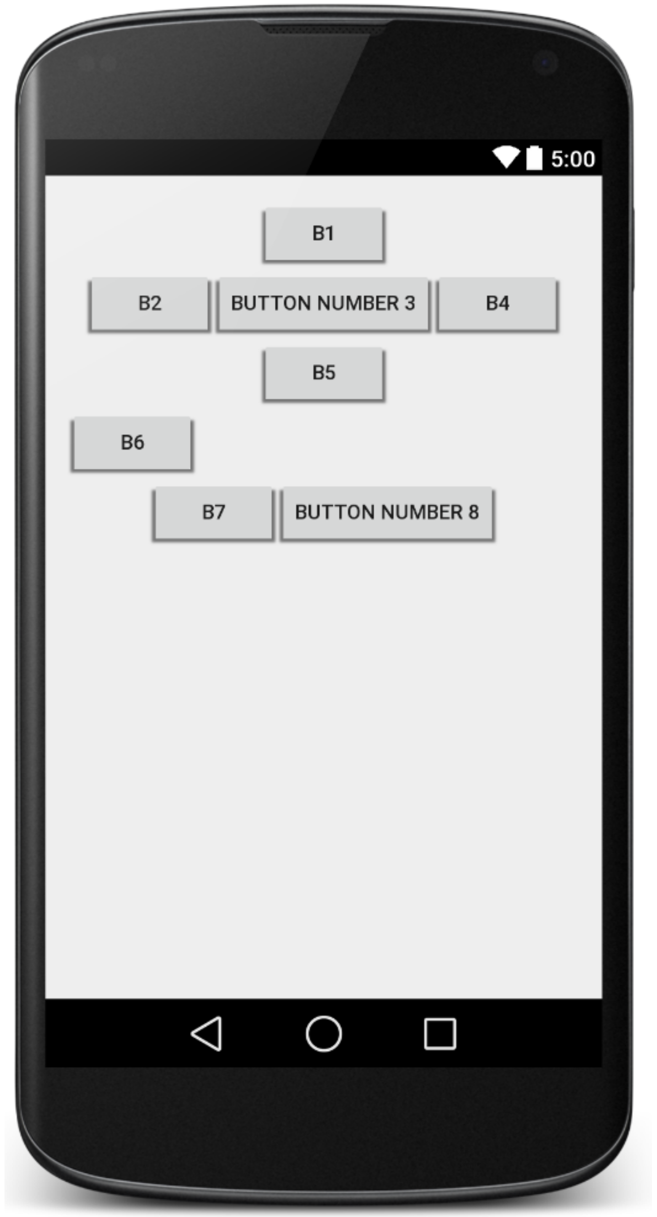
Nested layouts

- to produce more complicated appearance, use a **nested** layout

```
<OuterLayoutType ...>

    <InnerLayoutType ...>
        <Widget ... />
        <Widget ... />
    </InnerLayoutType>

    <InnerLayoutType ...>
        <Widget ... />
        <Widget ... />
    </InnerLayoutType>

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



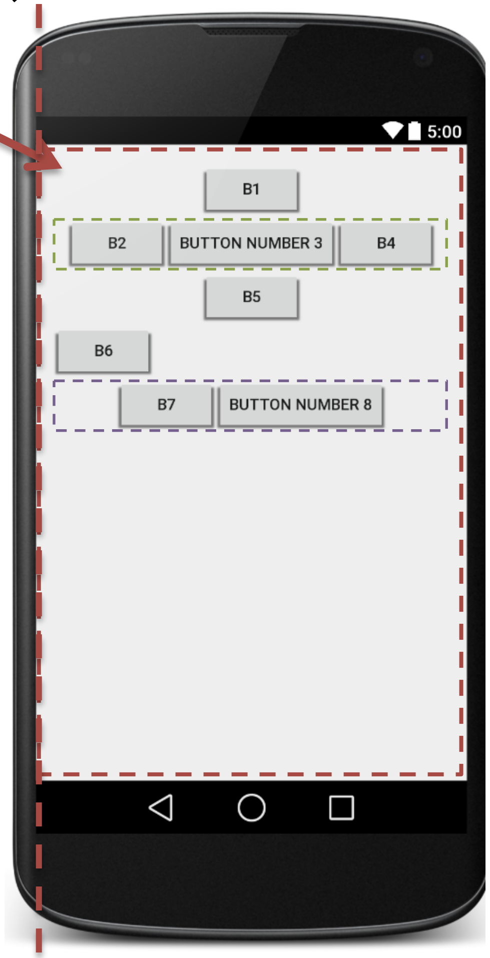
Nested layouts



?

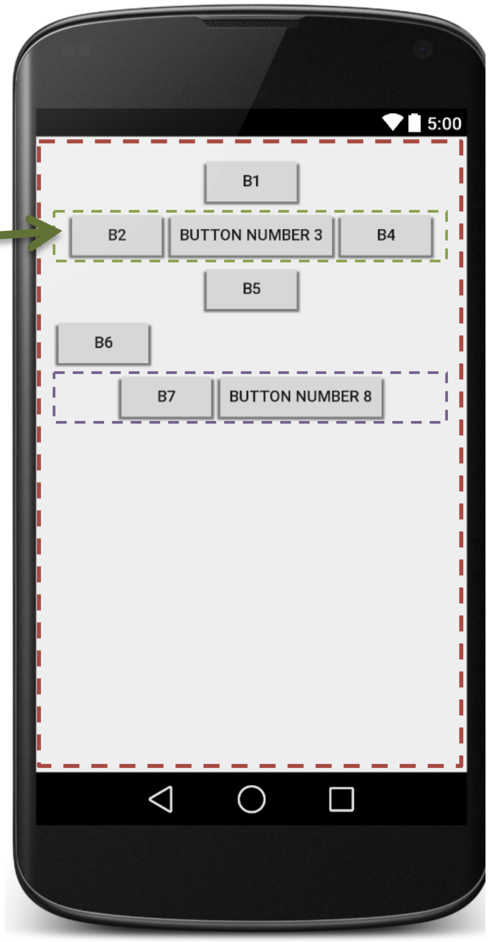
Nested layouts

```
<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>
```



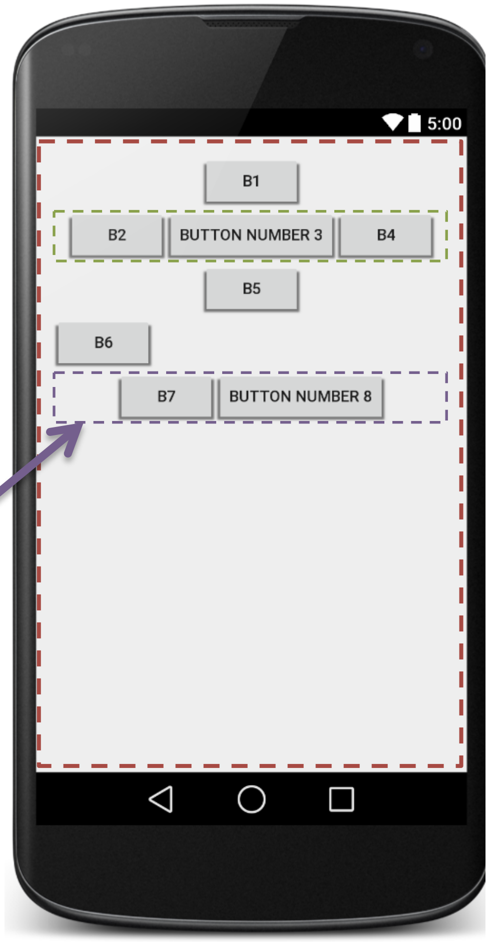
Nested layouts

```
<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>
```



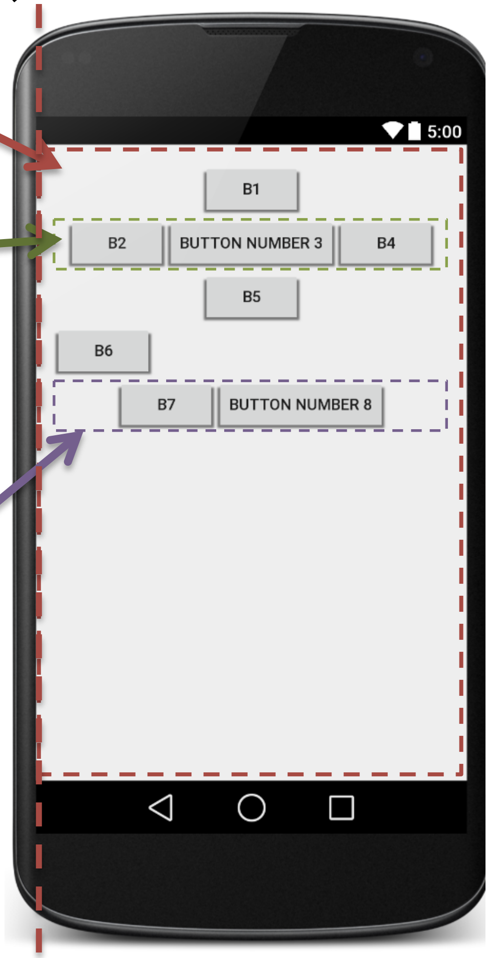
Nested layouts

```
<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>
```



Nested layouts

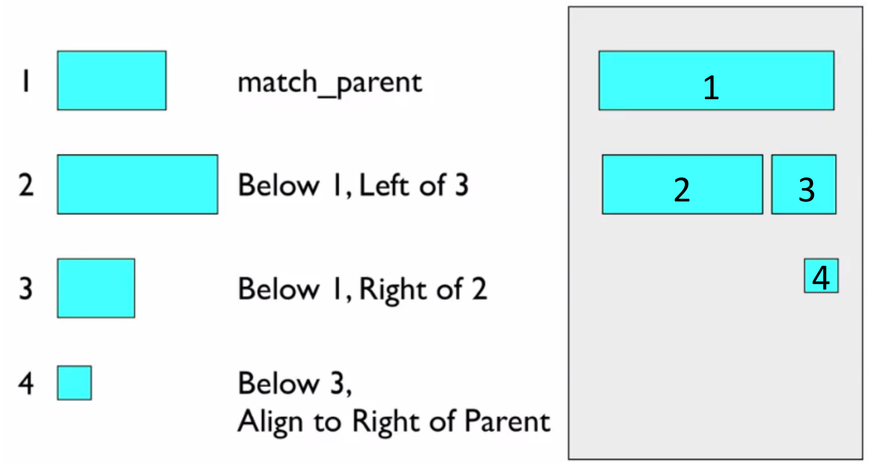
```
<LinearLayout ...
        android:orientation="vertical" android:gravity="center|top">
    <Button ... android:text="B1" />
    <LinearLayout ...
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
            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>
```



Relative Layout

RelativeLayout

- 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 layout

- Properties for x/y relative to **another widget**:

`layout_[below | above | toLeftOf | toRightOf]`

Positions this view [below, above…] the given view ID.

`layout_align[Baseline | Bottom| Left | Right | Top]`

Positions this view so that it is aligned the given view ID.

Example:

```
android:layout_below="@+id/button1”
android:layout_alignBottom="@+id/button3"
```

Relative layout

- Properties for x/y relative to layout **container** (the activity):

`layout_alignParent[Top | Bottom | Left | Right]`

Set these flags to a boolean value of "true" to enable them

`layout_center[Horizontal | Vertical | InParent]`

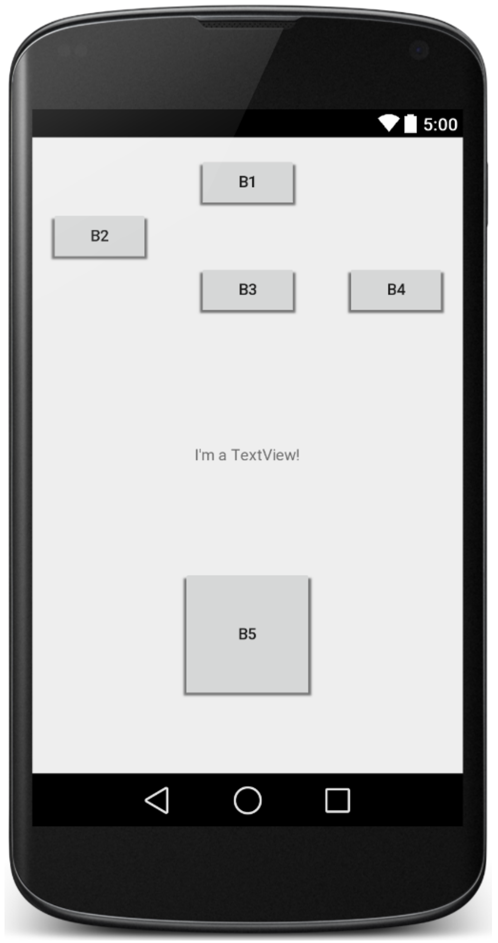
Set these flags to "true" to center the control within its parent in a dimension

- Example

```
android:layout_alignParentRight="true”
android:layout_centerInParent="true"
```

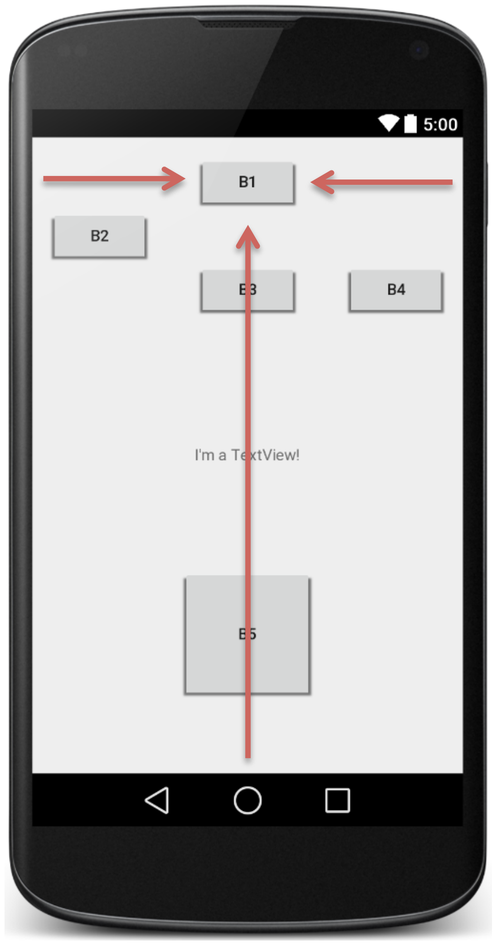
Example

```
<RelativeLayout ...>

    <Button ... android:id="@+id/button1” android:text="B1"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"/>

    <Button ... android:id="@+id/button2” android:text="B2"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/button1” />

    <Button... android:id="@+id/button3” android:text="B3"
        android:layout_alignLeft="@+id/button1"
        android:layout_below="@+id/button2” />

    <Button... android:id="@+id/button4” android:text="B4"
        android:layout_alignBaseline="@+id/button3"
        android:layout_alignBottom="@+id/button3"
        android:layout_alignParentRight="true”/>

    <TextView... android:id="@+id/textView1 »
           android:layout_centerInParent="true"
           android:text="I’m a TextView" />

    <Button... android:id="@+id/button5” android:text="B5"
        android:layout_alignLeft="@+id/button3"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="48dp”/>

</RelativeLayout>
```



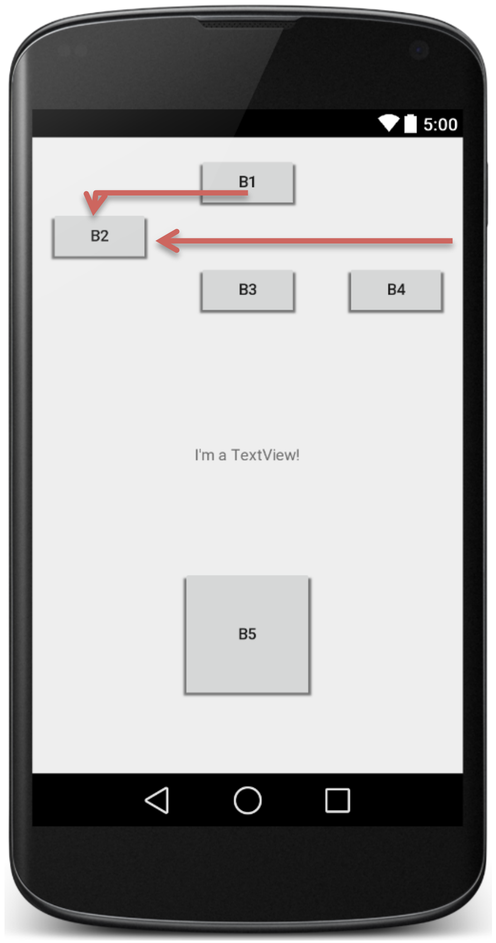
Example

```
<RelativeLayout ...>

  <Button ... android:id="@+id/button1” android:text="B1"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"/>

  <Button ... android:id="@+id/button2” android:text="B2"
      android:layout_alignParentLeft="true"
      android:layout_below="@+id/button1” />

  <Button... android:id="@+id/button3” android:text="B3"
      android:layout_alignLeft="@+id/button1"
      android:layout_below="@+id/button2” />

  <Button... android:id="@+id/button4” android:text="B4"
      android:layout_alignBaseline="@+id/button3"
      android:layout_alignBottom="@+id/button3"
      android:layout_alignParentRight="true”/>

  <TextView... android:id="@+id/textView1 »
         android:layout_centerInParent="true"
         android:text="I’m a TextView" />

  <Button... android:id="@+id/button5” android:text="B5"
      android:layout_alignLeft="@+id/button3"
      android:layout_alignParentBottom="true"
      android:layout_marginBottom="48dp”/>

</RelativeLayout>
```



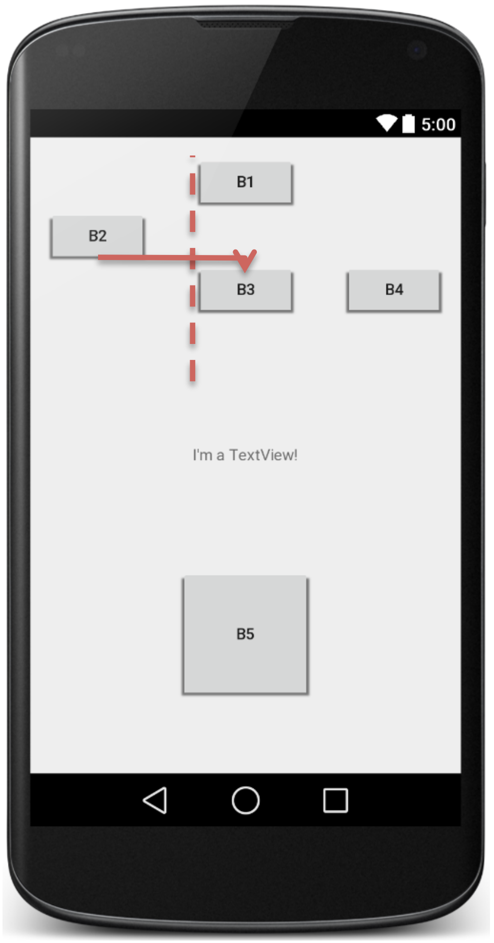
Example

```
<RelativeLayout ...>

    <Button ... android:id="@+id/button1” android:text="B1"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"/>

    <Button ... android:id="@+id/button2” android:text="B2"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/button1” />

    <Button... android:id="@+id/button3” android:text="B3"
        android:layout_alignLeft="@+id/button1"
        android:layout_below="@+id/button2” />

    <Button... android:id="@+id/button4” android:text="B4"
        android:layout_alignBaseline="@+id/button3"
        android:layout_alignBottom="@+id/button3"
        android:layout_alignParentRight="true”/>

    <TextView... android:id="@+id/textView1 »
           android:layout_centerInParent="true"
           android:text="I’m a TextView" />

    <Button... android:id="@+id/button5” android:text="B5"
        android:layout_alignLeft="@+id/button3"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="48dp”/>

</RelativeLayout>
```



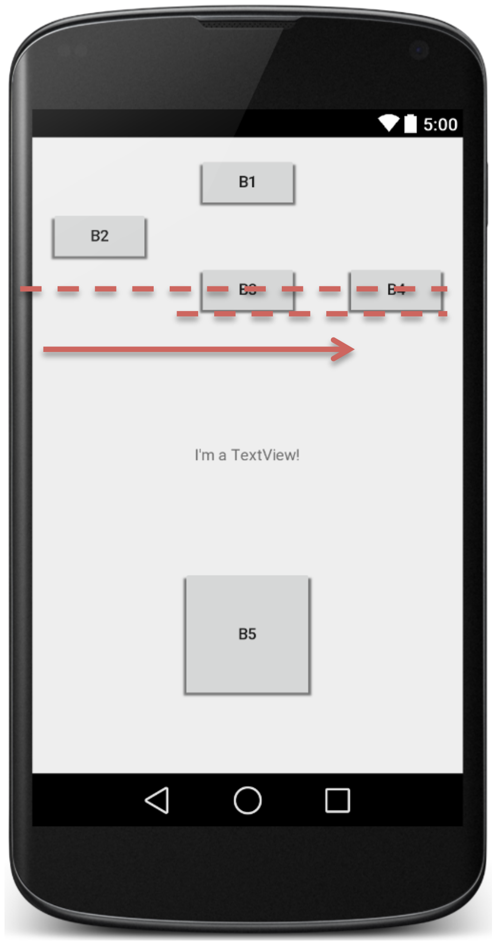
Example

```
<RelativeLayout ...>

    <Button ... android:id="@+id/button1” android:text="B1"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"/>

    <Button ... android:id="@+id/button2” android:text="B2"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/button1” />

    <Button... android:id="@+id/button3” android:text="B3"
        android:layout_alignLeft="@+id/button1"
        android:layout_below="@+id/button2” />

    <Button... android:id="@+id/button4” android:text="B4"
        android:layout_alignBaseline="@+id/button3"
        android:layout_alignBottom="@+id/button3"
        android:layout_alignParentRight="true”/>

    <TextView... android:id="@+id/textView1 »
           android:layout_centerInParent="true"
           android:text="I’m a TextView" />

    <Button... android:id="@+id/button5” android:text="B5"
        android:layout_alignLeft="@+id/button3"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="48dp”/>

</RelativeLayout>
```



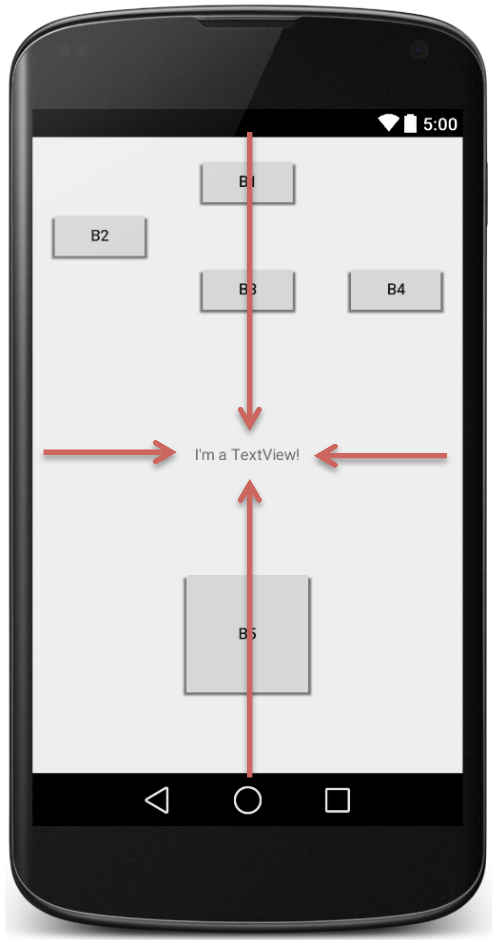
Example

```
<RelativeLayout ...>

    <Button ... android:id="@+id/button1” android:text="B1"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"/>

    <Button ... android:id="@+id/button2” android:text="B2"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/button1” />

    <Button... android:id="@+id/button3” android:text="B3"
        android:layout_alignLeft="@+id/button1"
        android:layout_below="@+id/button2” />

    <Button... android:id="@+id/button4” android:text="B4"
        android:layout_alignBaseline="@+id/button3"
        android:layout_alignBottom="@+id/button3"
        android:layout_alignParentRight="true”/>

    <TextView... android:id="@+id/textView1 »
           android:layout_centerInParent="true"
           android:text="I’m a TextView" />

    <Button... android:id="@+id/button5” android:text="B5"
        android:layout_alignLeft="@+id/button3"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="48dp”/>

</RelativeLayout>
```



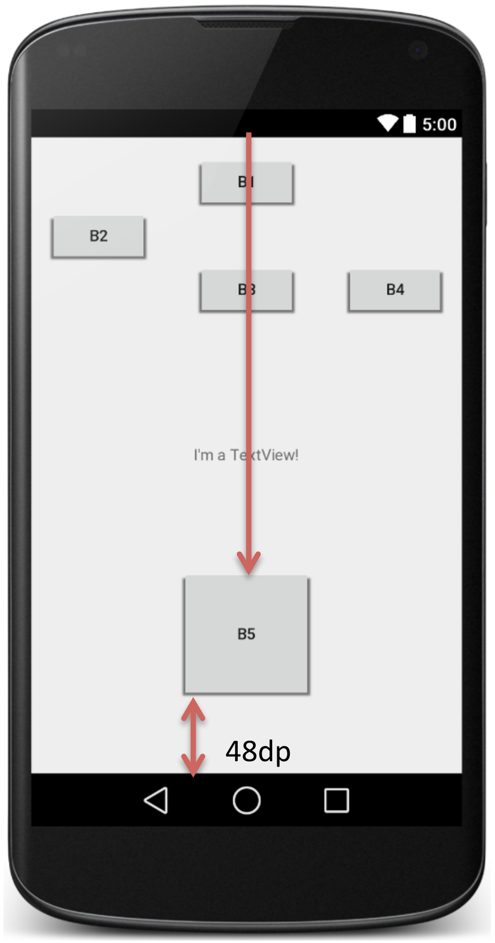
Example

```
<RelativeLayout ...>

    <Button ... android:id="@+id/button1” android:text="B1"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"/>

    <Button ... android:id="@+id/button2” android:text="B2"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/button1” />

    <Button... android:id="@+id/button3” android:text="B3"
        android:layout_alignLeft="@+id/button1"
        android:layout_below="@+id/button2” />

    <Button... android:id="@+id/button4” android:text="B4"
        android:layout_alignBaseline="@+id/button3"
        android:layout_alignBottom="@+id/button3"
        android:layout_alignParentRight="true”/>

    <TextView... android:id="@+id/textView1 »
           android:layout_centerInParent="true"
           android:text="I’m a TextView" />

    <Button... android:id="@+id/button5” android:text="B5"
        android:layout_alignLeft="@+id/button3"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="48dp”/>

</RelativeLayout>
```



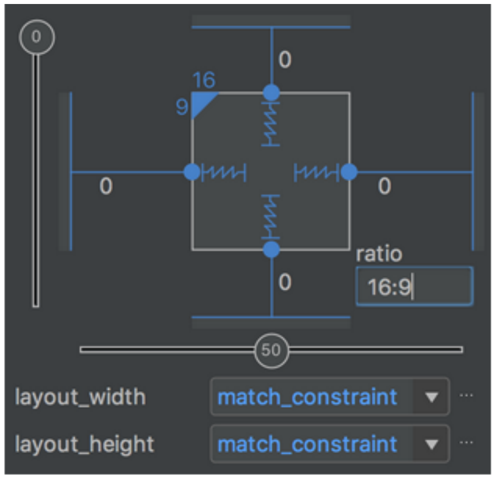
Example

```
<RelativeLayout ...>

    <Button ... android:id="@+id/button1” android:text="B1"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"/>

    <Button ... android:id="@+id/button2” android:text="B2"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/button1” />

    <Button... android:id="@+id/button3” android:text="B3"
        android:layout_alignLeft="@+id/button1"
        android:layout_below="@+id/button2” />

    <Button... android:id="@+id/button4” android:text="B4"
        android:layout_alignBaseline="@+id/button3"
        android:layout_alignBottom="@+id/button3"
        android:layout_alignParentRight="true”/>

    <TextView... android:id="@+id/textView1 »
           android:layout_centerInParent="true"
           android:text="I’m a TextView" />

    <Button... android:id="@+id/button5” android:text="B5"
        android:layout_alignLeft="@+id/button3"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="48dp”/>
</RelativeLayout>
```



Constrained layout

- Complex layouts with a flat view hierarchy (no nested view groups).
- Similar to RelativeLayout
  - All views are laid out according to relationships between sibling views and the parent layout,
  - more flexible than RelativeLayout and easier to use with AS's Layout Editor.
- Layout API and the Layout Editor were specially built for each other.
  - Just drag-and-dropping instead of editing the XML.
- Available in an API library that's compatible with Android 2.3 (API level 9) and higher.



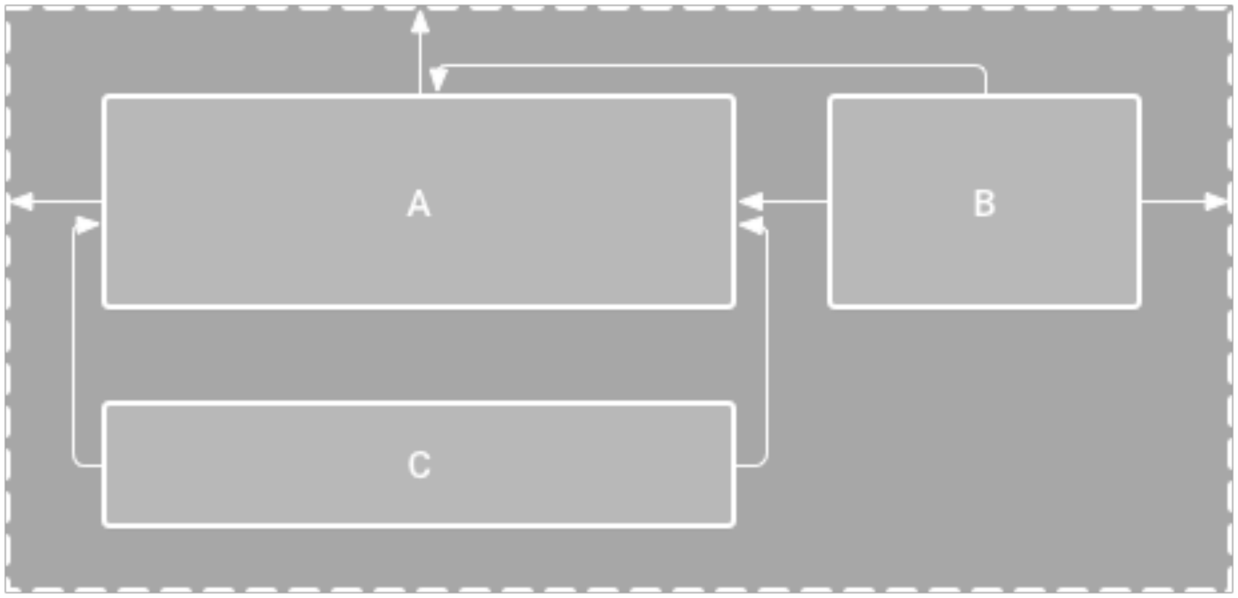
ratio 16:9
layout_width match_constraint
layout_height match_constraint

Constrained layout

- To define a view's position in ConstraintLayout, you must add at least one horizontal and one vertical constraint for the view.
- Each constraint represents a connection or alignment to another view, the parent layout, or an invisible guideline.
- Each constraint defines the view's position along either the vertical or horizontal axis; so each view must have a minimum of one constraint for each axis, but often more are necessary.
- When you drop a view into the Layout Editor, it stays where you leave it even if it has no constraints.
  - this is only to make editing easier; if a view has no constraints when you run your layout on a device, it is drawn at position [0,0] (the top-left corner).

Constrained layout

- Example: each view must have a minimum of one constraint for each axis, but often more are necessary.

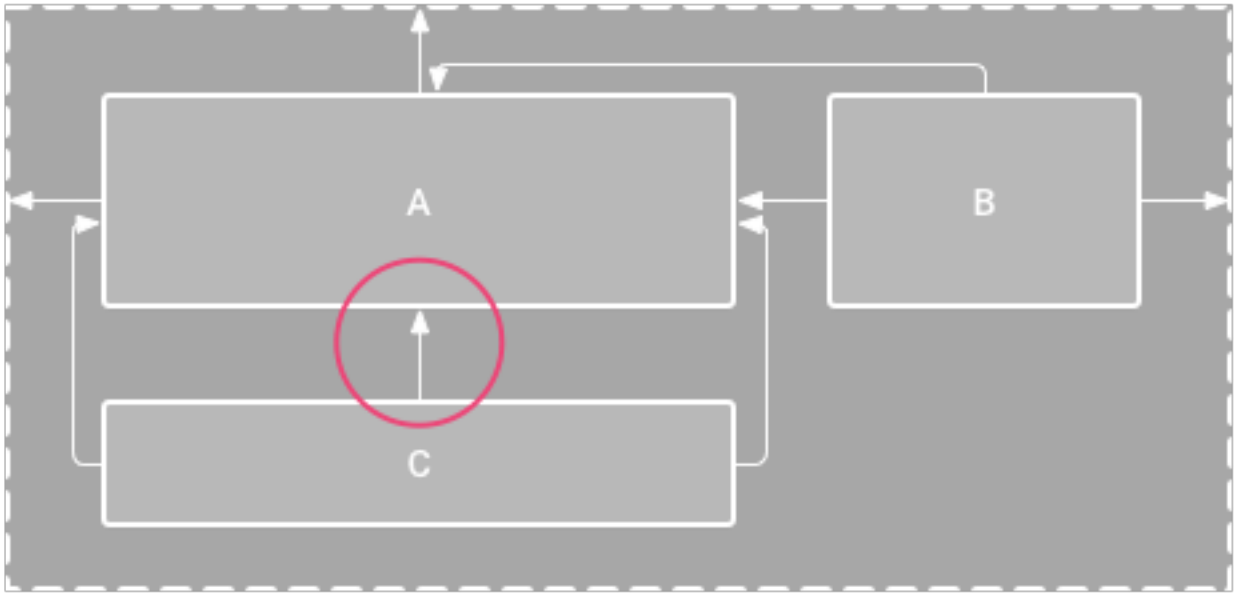


C has no vertical constraint

When this layout draws on a device, view C horizontally aligns with the left and right edges of view A, but appears at the top of the screen because it has no vertical constraint.

Constrained layout

- Example: each view must have a minimum of one constraint for each axis, but often more are necessary.

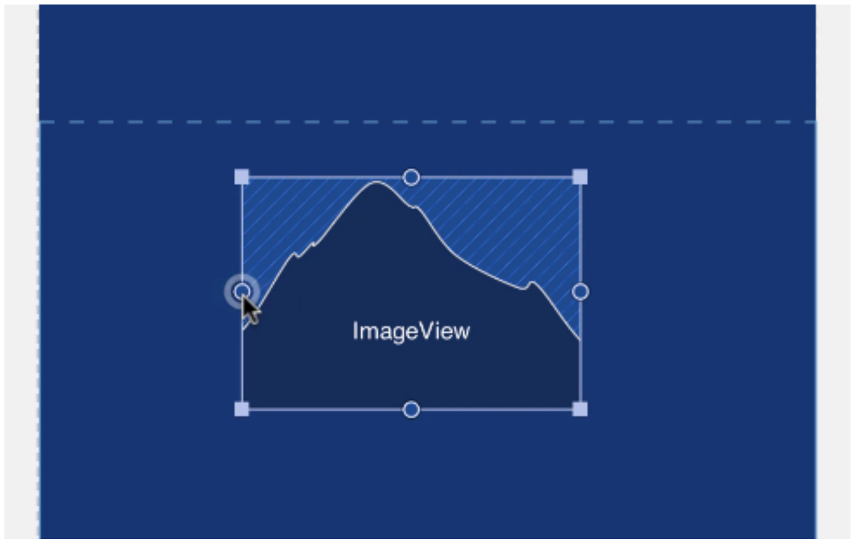


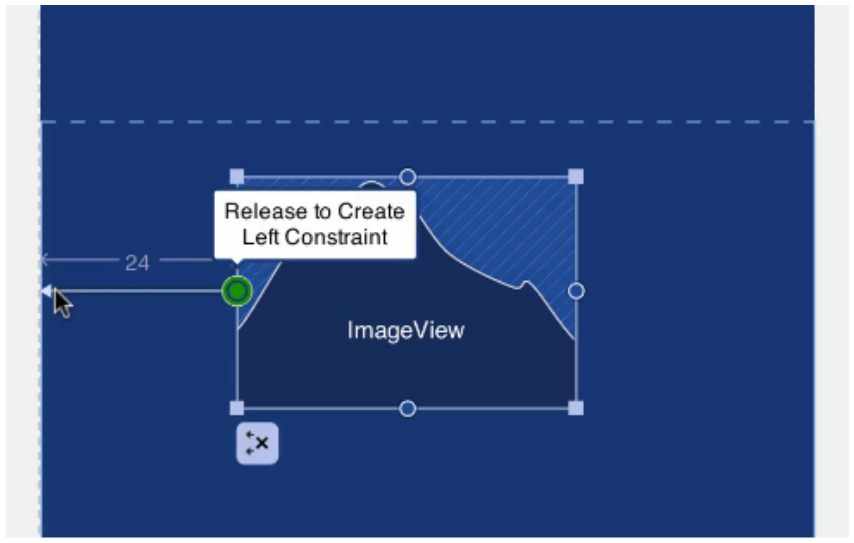
View C is now vertically constrained below view A

Constrained layout

To add a constraint do one of the following:

- Click a constraint handle and drag it to an available anchor point
  - the edge of another view,
  - the edge of the layout,
  - or a guideline.

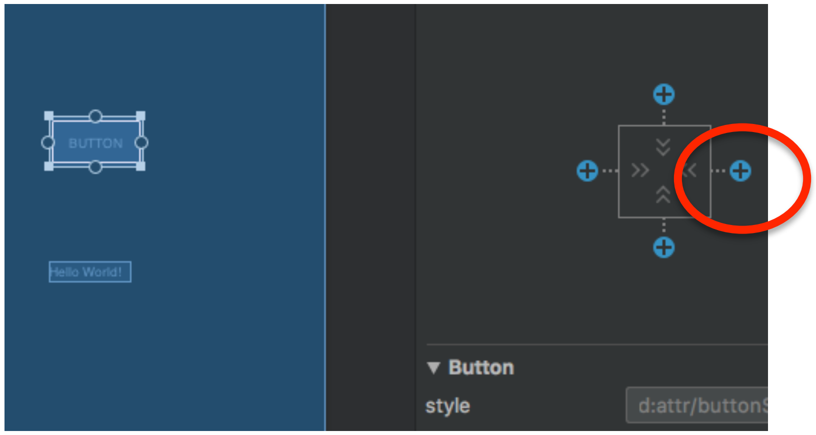


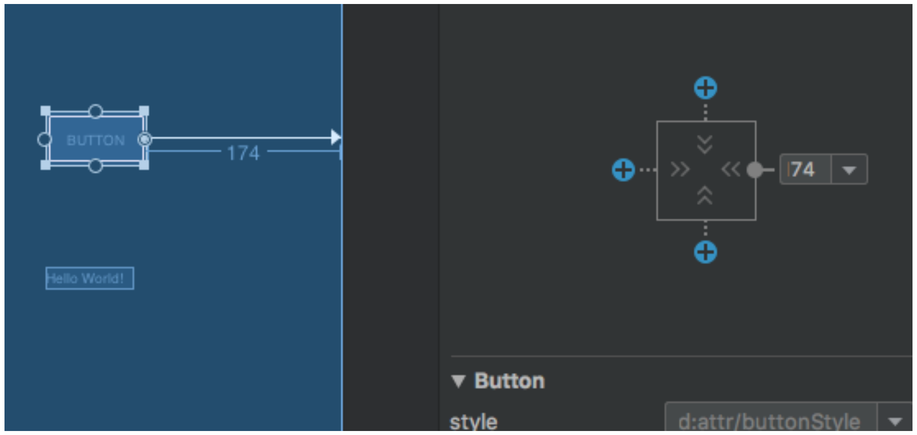


Constrained layout

To add a constraint do one of the following:

- Click **Create a connection** in the view inspector at the top of the **Attributes** window.





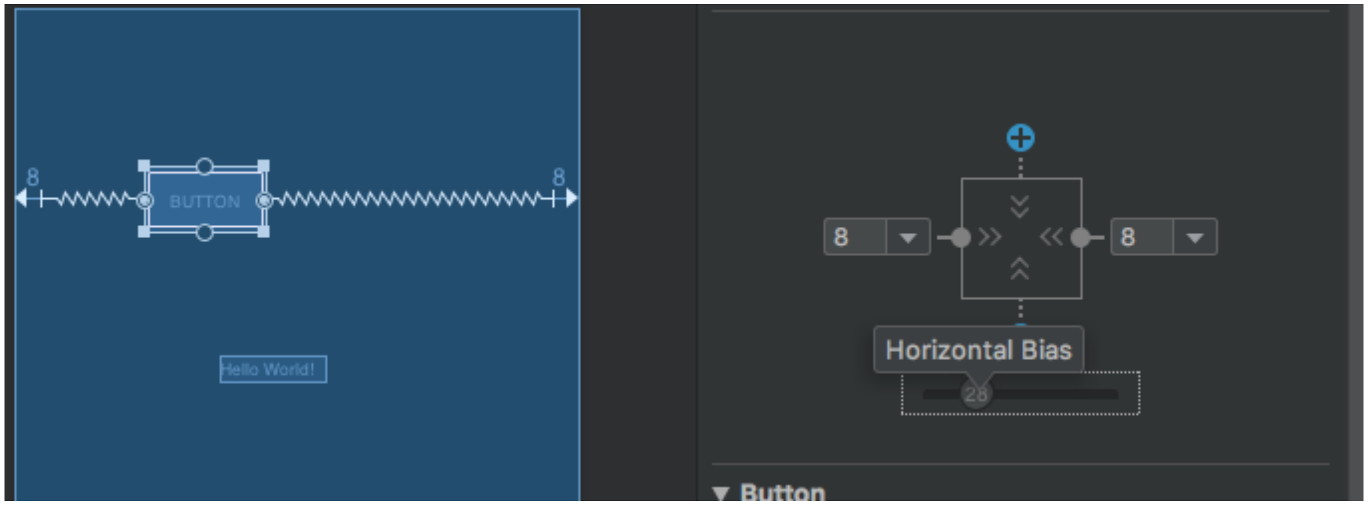
Constrained layout

When creating constraints, remember the following rules:

- Every view must have **at least two constraints**: one horizontal and one vertical.
- You can create constraints only between a constraint handle and an anchor point **that share the same plane**. So a vertical plane (the left and right sides) of a view can be constrained only to another vertical plane; and baselines can constrain only to other baselines.

Constrained layout

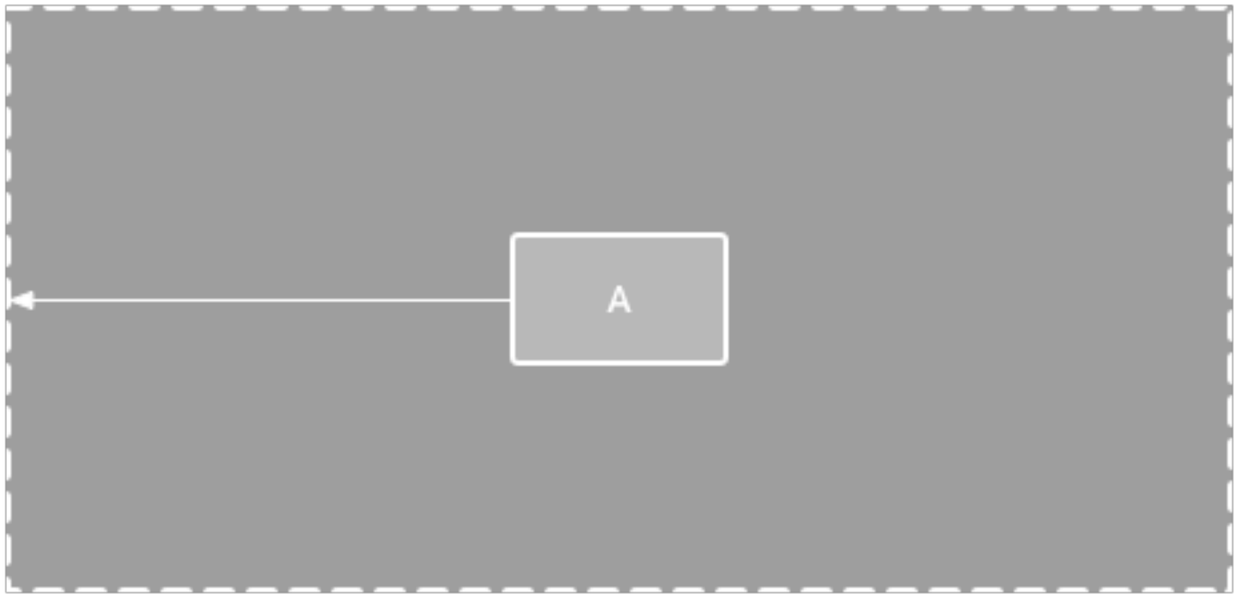
- If you add opposing constraints on a view, the constraint lines become like a spring to indicate the opposing forces
- The view is centered between the constraints.
- If you want to move the view so that it is not centered, adjust the **constraint bias**.



Constrained layout

Parent position

- Constrain the side of a view to the corresponding edge of the layout.

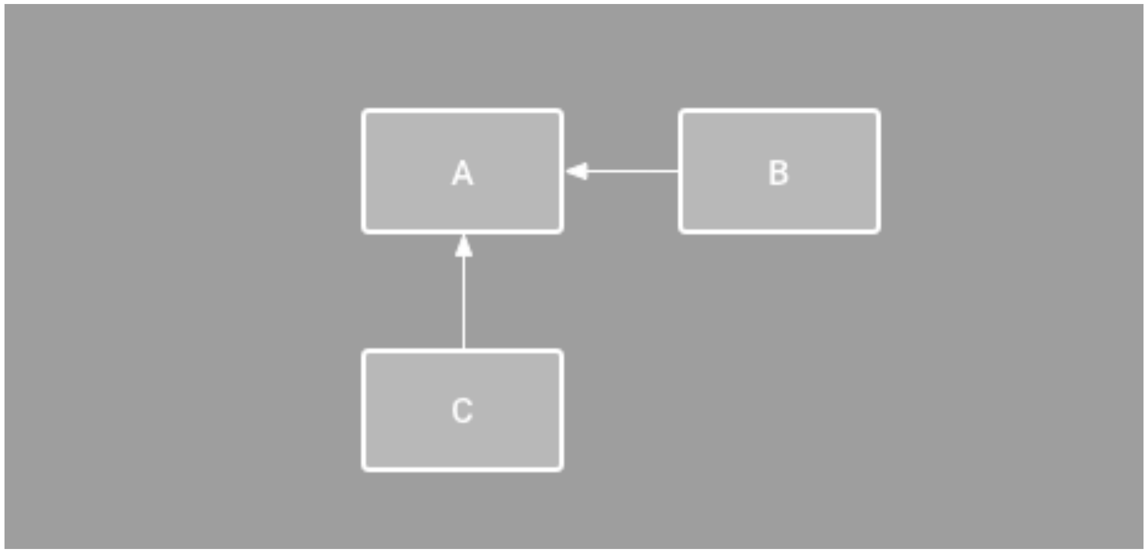


the left side of the view A is connected to the left edge of the parent layout. You can define the distance from the edge with margin.

Constrained layout

Order position

- Define the order of appearance for two views, either vertically or horizontally.

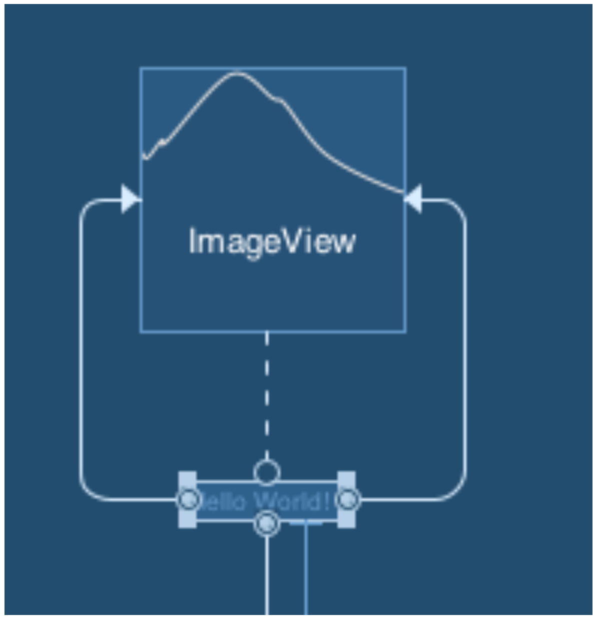


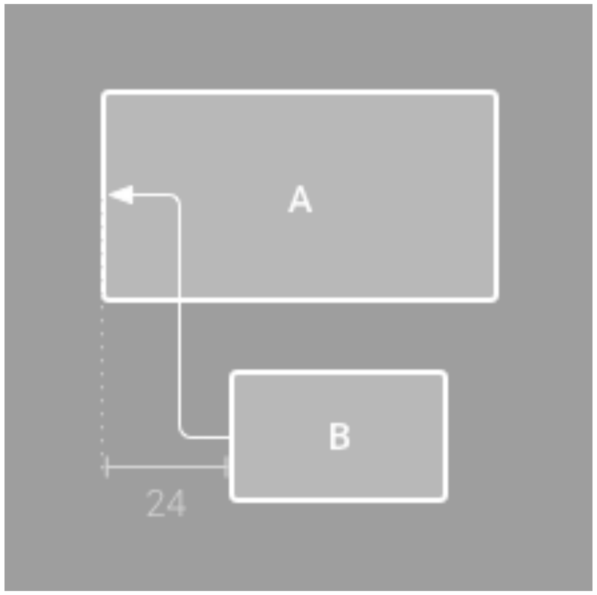
- B is constrained to always be to the right of A, and C is constrained below A. However, these constraints do not imply alignment, so B can still move up and down.

Constrained layout

Alignment

- Align the edge of a view to the same edge of another view.





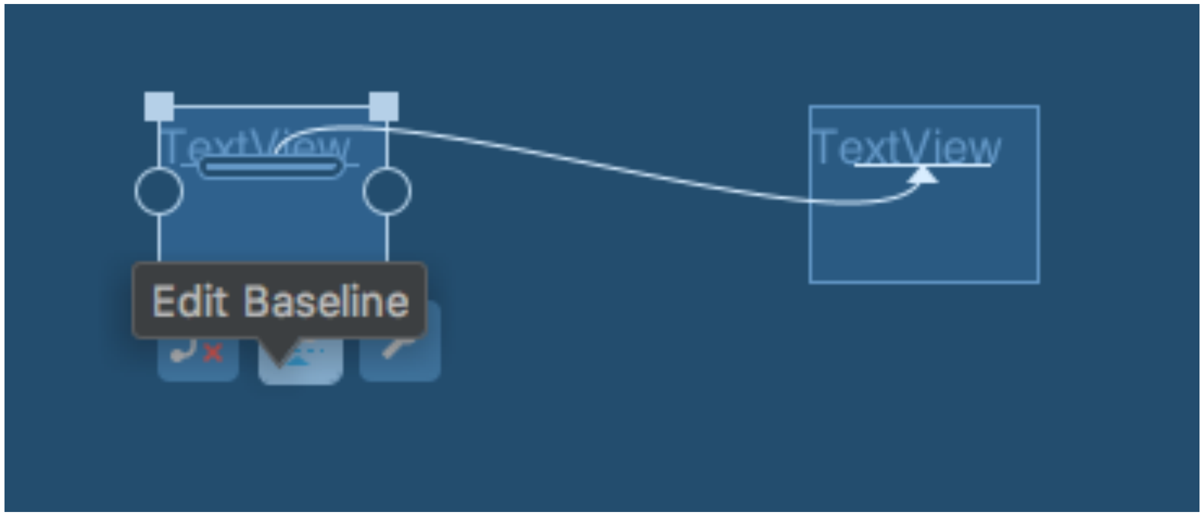
the left side of B is aligned to the left side of A.
To align the view centers, create a constraint on both sides.

To offset the alignment drag the view inward from the constraint.
B with a 24dp offset alignment.

Constrained layout

Baseline alignment

- Align the text baseline of a view to the text baseline of another view.



- The first line of B is aligned with the text in A.
- To create a baseline constraint, select the text view you want to constrain and then click **Edit Baseline** , which appears below the view. Then click the text baseline and drag the line to another baseline.

Plan

- UI as a resource
- UI elements
  - Widgets
  - Layouts
    - Linear
    - Grid
    - Relative
    - Constrained
- The API

The API

- The user interface for an activity is provided by a hierarchy of objects derived from the `View` class.
- **Widgets** are all the visual and interactive elements
  - button, text field, checkbox etc..
- **Layouts** provide a unique layout model for its child views,
  - linear layout, a grid layout, or relative layout.

http://developer.android.com/guide/components/activities.html#UI

Load the GUI

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

Load the GUI elements from the resource class

Get GUI elements

`findViewById()` to get the reference to each GUI element

```
setContentView( R.layout.activity_main );

TextView mText = (TextView) findViewById( R.id.textbox );
```

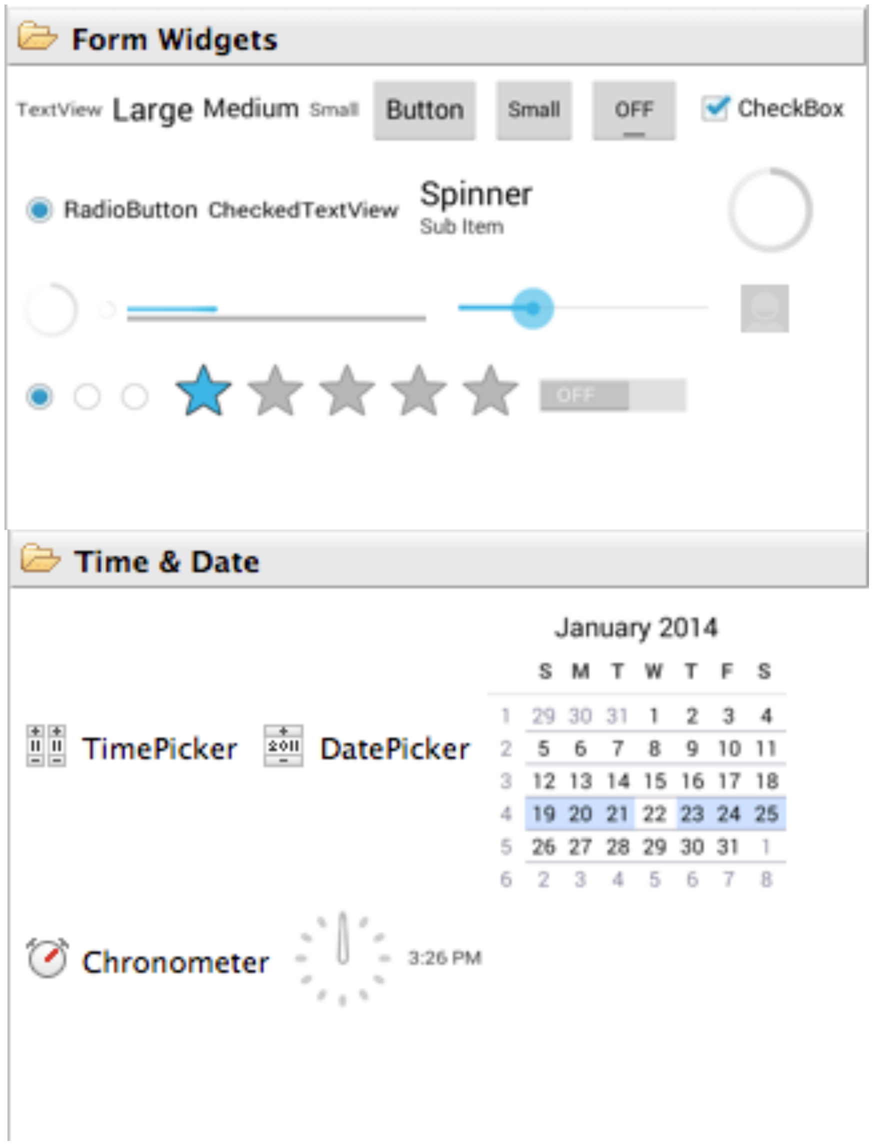
Starting from API 26, no cast required: `TextView mText = findViewById( R.id.textbox );`

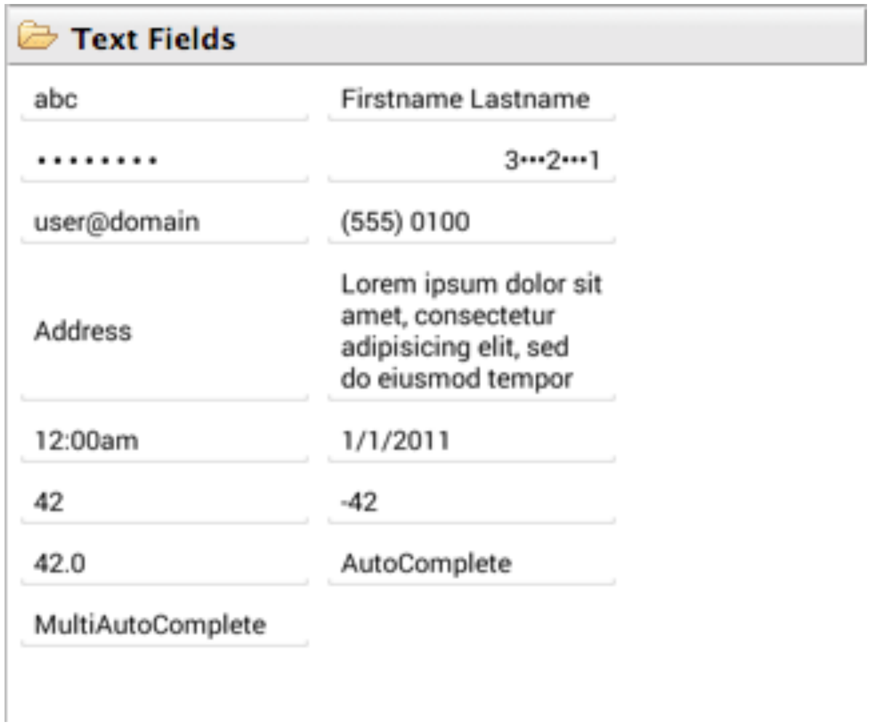
```
…
    <TextView
        android:id="@+id/textbox"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" >
     </TextView>

</RelativeLayout>
```

activity_main.xml

- Remember to cast the result of
- mText now points to the object and you can modify it

Android widgets





And many more!

Button

A clickable widget with a text label

`android:clickable="`***bool***`”`	set to false to disable the button
`android:id="@+id/`***theID***`”`	unique ID for use in Java code
`android:onClick="`***function***`”`	function to call in activity when clicked (must be public, void, and take a View arg)
`android:text="`***text***`”`	text to put in the button (**USE STRINGS!**)

represented by Button class in Java code

```
Button b = (Button) findViewById( R.id.theID )
```

Setting up the onClick() callback

- **From the XML file**
  - Set the `android:onClick` attribute to a public function of the **activity**

```
 <Button
        android:id="@+id/button1”
        android:onClick="buttonClick"
        android:text="Button" />
```

Activity_main.xml

```
public class MainActivity extends Activity
{
    ...
    public void buttonClick(View v)
    {
        // deal with the button action here

    }
```

MainActivity.java

Setting up the onClick() callback

- **At run-time**
  1. Define a callback implementing `View.onClickListener`
  2. Get the button and set the callback with its `setOnClickListener` method

With an anonymous class (1)

```
Button b = (Button) findViewById( R.id.button1 );
b.setOnClickListener( new View.OnClickListener()
{

    @Override
    public void onClick( View v )
    {
        // TODO Auto-generated method stub

    }
} );
```

Setting up the onClick() callback

- **At run-time**
  1. Define a callback implementing `View.onClickListener`
  2. Get the button and set the callback with its `setOnClickListener` method

With an anonymous class (2)

```
public class MainActivity extends Activity
{
     private View.OnClickListener buttonLis = new View.OnClickListener()
     {
          @Override
          public void onClick( View v )
          {
               // your code here
          }
     };

     @Override
     protected void onCreate( Bundle savedInstanceState )
     {
          ...
          Button b = (Button) findViewById( R.id.button1 );
          b.setOnClickListener( buttonLis);
```

Setting up the onClick() callback

- **At run-time**
  1. Define a callback implementing `View.onClickListener`
  2. Get the button and set the callback with its `setOnClickListener` method

Activity implements the listener interface

```
public class MainActivity extends Activity implements View.OnClickListener
{
     @Override
     public void onClick( View v )
     {
          // your code here
     }

     @Override
     protected void onCreate( Bundle savedInstanceState )
     {
          ...
          Button b = (Button) findViewById( R.id.button1 );
          b.setOnClickListener( this);
```

Setting up the onClick() callback

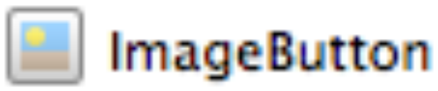
- Using a single function to deal with many buttons
  - `android:onClick`
  - Anonymous class (2)
  - Activity implements interface

```
@Override
public void onClick( View v )
{
    if( v.getId() == R.id.button1 )
    {
        // code for button1
    }
    else if( v.getId() == R.id.button2 )
    {
        // code for button2
    }
    else if( v.getId() == R.id.button3 )
    {
        // code for button3
    }
```

Or use **`switch`**

ImageButton

A clickable widget with an image label



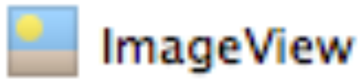
`android:clickable="`***bool***`”`	set to false to disable the button
`android:id="@+id/`***theID***`”`	unique ID for use in Java code
`android:onClick="`***function***`”`	function to call in activity when clicked (must be public, void, and take a View arg)
`android:src=”`***@drawable/img***`”`	Image for the button, it must correspond to an image resource

to set up an image resource:

- put image file in project folder **app/src/main/res/drawable**
- use @drawable/foo to refer to foo.png

ImageView

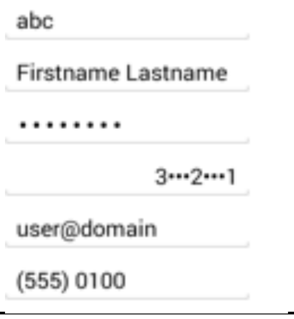
A clickable widget with an image label



`android:id="@+id/`***`theID`***`”`	unique ID for use in Java code
`android:src=”`***`@drawable/img`***`”`	Image for the button, it must correspond to an image resource

EditText

A clickable widget with an image label


abc
Firstname Lastname
....
3•••2•••1
user@domain
(555) 0100

`android:hint="text"`	Grey text to show before user input
`android:inputType="`***`type`***`"`	The type of input to be typed (number, mail...)
`android:id="@+id/`***`theID`***`"`	unique ID for use in Java code
`android:lines="`***`int`***`"`	Number of visible lines
`android:maxLines="`***`int`***`"`	Max number of lines that the user can enter

RadioButton

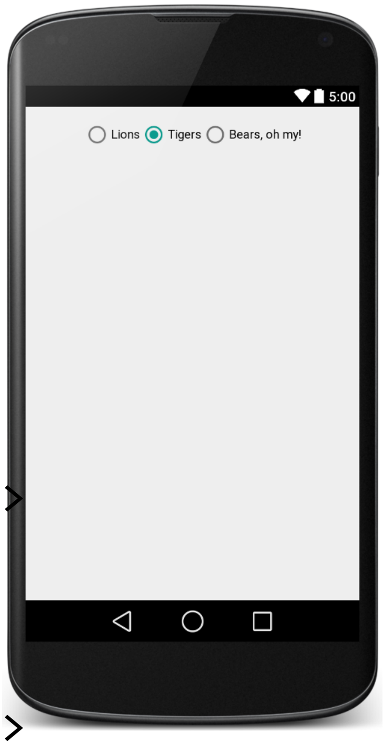
A toggleable on/off switch; part of a group

`android:clickable="`***`bool`***`”`	set to false to disable the button
`android:checked="`***`bool`***`”`	set to true to have it checked at the beginning
`android:id="@+id/`***`theID`***`”`	unique ID for use in Java code
`android:onClick="`***`function`***`”`	function to call in activity when clicked (must be public, void, and take a View arg)
`android:text=”`***`text`***`”`	Text to place close to the button

need to be nested inside a RadioGroup tag in XMLso that only one can be selected at a time

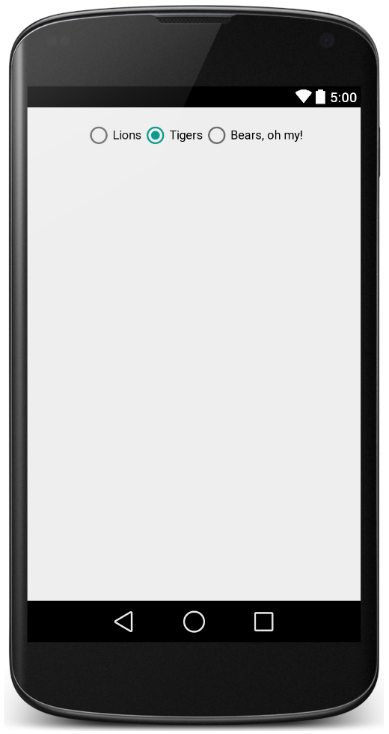
RadioButton

```
<LinearLayout ...
        android:orientation="vertical"
        android:gravity="center|top">
    <RadioGroup ...
            android:orientation="horizontal">
        <RadioButton ... android:id="@+id/lions"
                         android:text="Lions"
                         android:onClick="radioClick" />
        <RadioButton ... android:id="@+id/tigers"
                         android:text="Tigers"
                         android:checked="true"
                         android:onClick="radioClick" />
        <RadioButton ... android:id="@+id/bears"
                         android:text="Bears, oh my!"
                         android:onClick="radioClick" />
    </RadioGroup>
</LinearLayout>
```



RadioButton

```
// in MainActivity.java
public class MainActivity extends Activity {

    public void radioClick(View view) {
        // check which radio button was clicked
        if (view.getId() == R.id.lions) {
            // ...
        } else if (view.getId() == R.id.tigers) {
            // ...
        } else {
            // bears ...
        }
    }
```



Spinner

A drop-down menu of selectable choices

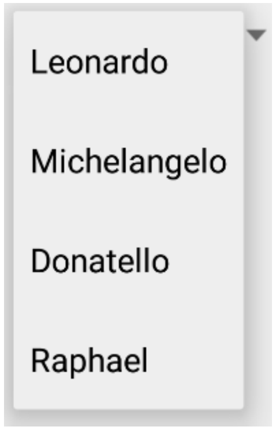
`android:clickable="`***`bool`***`”`	set to false to disable the spinner
`android:id="@+id/`***`theID`***`”`	unique ID for use in Java code
`android:entries=”@array/`***`array`***`”`	Set of values to display (an array in `strings.xml`)
`android:prompt=`***`”@string/text”`***	Title text when the dialog of choices pops up

It needs to handle events in Java code

- must get the Spinner object using `findViewById()`
- then call its `setOnItemSelectedListener` method

Spinner

```
<LinearLayout ...>
    <Spinner ... android:id="@+id/tmnt"
        android:entries="@array/turtles"
        android:prompt="@string/choose_turtle" />
    <TextView ... android:id="@+id/result" />
</LinearLayout>
```



res/values/strings.xml

```
 <resources>
        <string name="choose_turtle">Choose a turtle:</string>
        <string-array name="turtles">
            <item>Leonardo</item>
            <item>Michelangelo</item>
            <item>Donatello</item>
            <item>Raphael</item>
        </string-array>
    </resources>
```

String arrays

```
<resources>
        <string name="name">value</string>
        <string name="name">value</string>

        <string-array name="arrayname">
            <item>value</item>
            <item>value</item>
            <item>value</item
            ...
            <item>value</item>
        </string-array>
    </resources>
```

References

- Part of these slides are readapted from **CS 193A, Lecture 2** by Marty Stepp

http://web.stanford.edu/class/cs193a/lectures/02-layout-gui.pdf