Layouts In Android

Mobile Computing - Android

Terminology

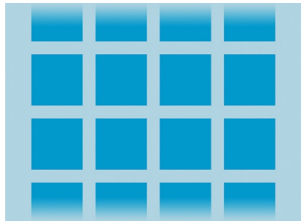
- “Layout” has 2 meanings in Android:
  1. An xml file, found in /res/layout - lays out the UI
  2. A ViewGroup subclass (LinearLayout, RelativeLayout, GridLayout, TableLayout, etc.)
- The former usually includes an xml tag that refers to the latter ... we'll see this in a moment.

Basics

- Linear - Stack of components

- Constraint – Placement related to other components or the parent.

- Grid - A grid of components



Legacy

- There are some layouts that were labeled as Legacy about 5 years ago. Two to be aware of are:
  - **RelativeLayout** –Up until the switch, this was the default layout used for many of the project templates. It was replaced by the more consistent ConstraintLayout. You might still see relative layouts it in older projects, but it is not recommended now.
  - **GridLayout** – This layout was also moved to legacy, but is still useful. It is similar to the CCS grid, while its replacement (TableLayout/TableRow) is similar to the basic HTML elements. We will learn grid in this course and you can try out the new layout on your own.

FrameLayout

- The most basic of ViewGroups.
- One drawable area where all components are drawn from first to last.
- Can use the positioning mechanism gravity, to slide components to the edges, but best to only have a single visible component.
- We will use it later with fragments.

Original Java

- Some of these layouts are close to the original java circa 1995 AWT (Abstract Window Toolkit) versions.
  - BorderLayout
  - FlowLayout
  - GridLayout

- Some of the components (widgets) are similar
- Some of the underlying classes (Color) are still in use.

XML

- Starts with <?xml version="1.0" encoding="utf-8"?>
- Since this is XML, each Layout file must contain 1 root element, a **View** (anything visible on the screen) or **ViewGroup** (a View that can contain other Views. Examples include LinearLayouts, ConstraintLayouts and GridLayouts). In nearly all circumstances, it will be a view group of some kind..
- We add additional ViewGroups or widgets (Views such as TextViews, EditTexts,Buttons, etc.) as *children*
- XML *element names* correspond directly to *class names* (e.g., <TextView> => android.widget.TextView; <Button> => android.widget.Button; <LinearLayout> ==> android.widget.LinearLayout)
- XML *attributes* correspond directly to *methods* (e.g., android:text="hello" corresponds to setText("hello"))
- Attributes may be:
  - specific to one View object (e.g., android:text="Name:")
  - common to all Views (e.g., android:id="@+id/nameET)
  - layout parameters (e.g., android:layout_width="wrap_content")
- Attributes generally start with word "android:" to avoid name clashes

Connections Reference/ResId/ XML name

```
public void clickToIncrease(View v){

    EditText sizeET = findViewById(R.id.sizeET);
    // access the EditText with id R.id.sizeET

    try {
        Double size = Double.parseDouble(sizeET.getText().toString());
        // getText() returns an Editable, not a String
        // parse may fail
        size += 1.0;
        sizeET.setText(size.toString);
    } catch (NumberFormatException ex) {
        Log.d(“Input”, “Unable to convert string into a float”)
    }
}
```

```
<EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/sizeET"
        android:hint=”Enter a size"
/>
```

In playground.xml

```
<?xml version="1.0" encoding="utf-8"?>

<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/headlinerTV"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Layout Playground"
        android:textAppearance="?android:attr/textAppearanceLarge"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0" />

    <EditText
        android:id="@+id/nameET"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:hint="Last Name, First Name"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/headlinerTV"
        app:layout_constraintVertical_bias="0.1" />
```

Root

Child

Child

```
    <TextView
        android:id="@+id/nameTV"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Name"
        android:textAppearance="?android:attr/textAppearanceSmall"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/nameET"
        app:layout_constraintVertical_bias="0.2" />


     <Button
        android:id="@+id/clickForProgressBTN"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:onClick="clickForProgress"
        android:text="Click For Progress"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/nameET"
        app:layout_constraintVertical_bias="0.3" />


</android.support.constraint.ConstraintLayout>
```

Child

Child

Loading XML

- View don't just magically appear: they need to be asked to appear by an activity
- Since layout files are resources, they are assigned a resource ID (specified in R), of the form R.layout.*layoutFileName*
- Use setContentView(), passing in that resource ID. This triggers the creation of the view from the XML resource. (Inflation)

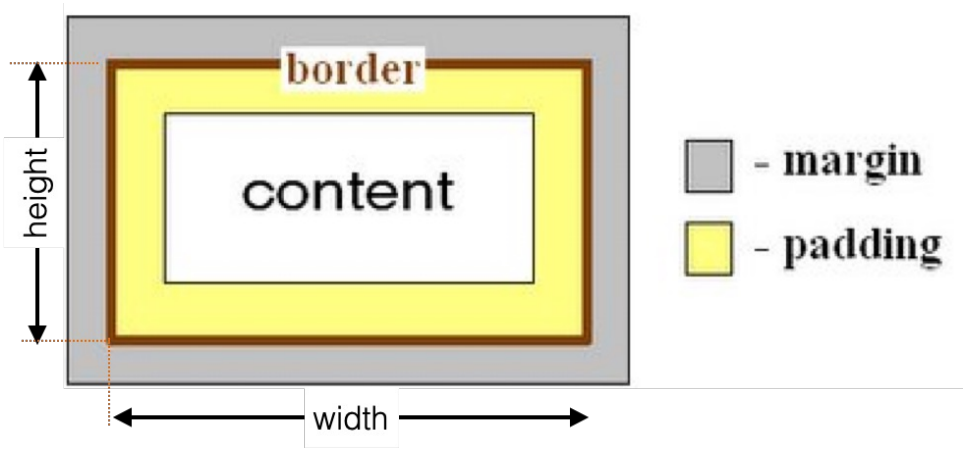
```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.playground);
}
```

View Location, Size

- Each View *must* specify its **layout_width** & **layout_height**
  - wrap_content, match_parent, or number (e.g. 350dp)
- Each View is defined by a rectangle (left, top, width, height)
- There are two widths & heights kept by a View:
  - getMeasuredWidth(), getMeasuredHeight() return *desired* width and height
  - **getWidth()** and **getHeight()** return *actual* width and height. These are usually what we need.

Padding & Margins

- Same basic idea as CSS
- Widgets have a **size**, specified by their width & height
- Widgets have **margins** - space between them and their neighbor(s)
  - Specify margins with android:layout_margin, orandroid:layout_marginStart, android:layout_marginEnd, etc.
- Widgets have **padding** - space between their edges and the actual content
  - Pad a View (compressing its content) by setting android:padding, or android:layout_paddingStart, android:layout_paddingEnd, etc.



Android Fun Fact # 37205:

android:layout_margin = "value" sets all 4 margins at once
android:padding = "value" sets all 4 padding values at once

Left/Right vs Start/End

Older versions of Android used left/right when talking about the horizontal bounds of a view. The modern way of specifying the bounds is to use start and end. For example:

- **Old Style**

  android:layout_marginLeft

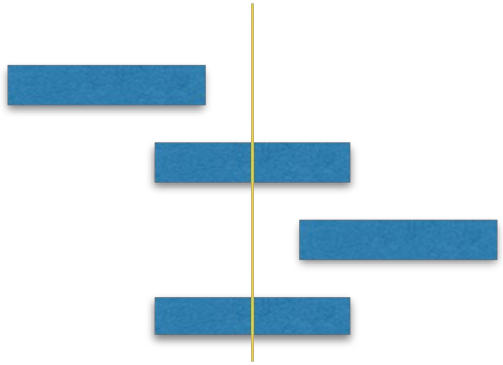
- **New Style**

  android:layout_marginStart

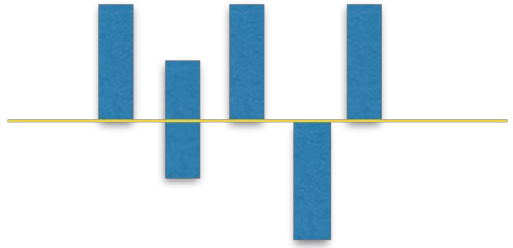
  The old style is still available, but the new style is preferred for better performance in the layout.

Linear Layouts

- All views line up, either stacked up on top one another (vertically) or in a row (horizontally)
- **android:orientation="vertical"**
  - android:layout_gravity="start|center_horizontal|end"
    - if objects are oriented vertically, they can float to the left, center, or right



- **android:orientation="horizontal"**
  - android:layout_gravity="top|center_vertical|bottom"
    - if objects are oriented horizontally, they can float to the top, center, or bottom



- Uses an "elder child" approach — the eldest child get what its want; if there is room left over, the next eldest child get what its want; if there is room left over, the next eldest child gets what its want, etc. Implication: if the first View asks to match_parent (height for vertical layouts; width for horizontal ones), there will be no room left over for other Views.

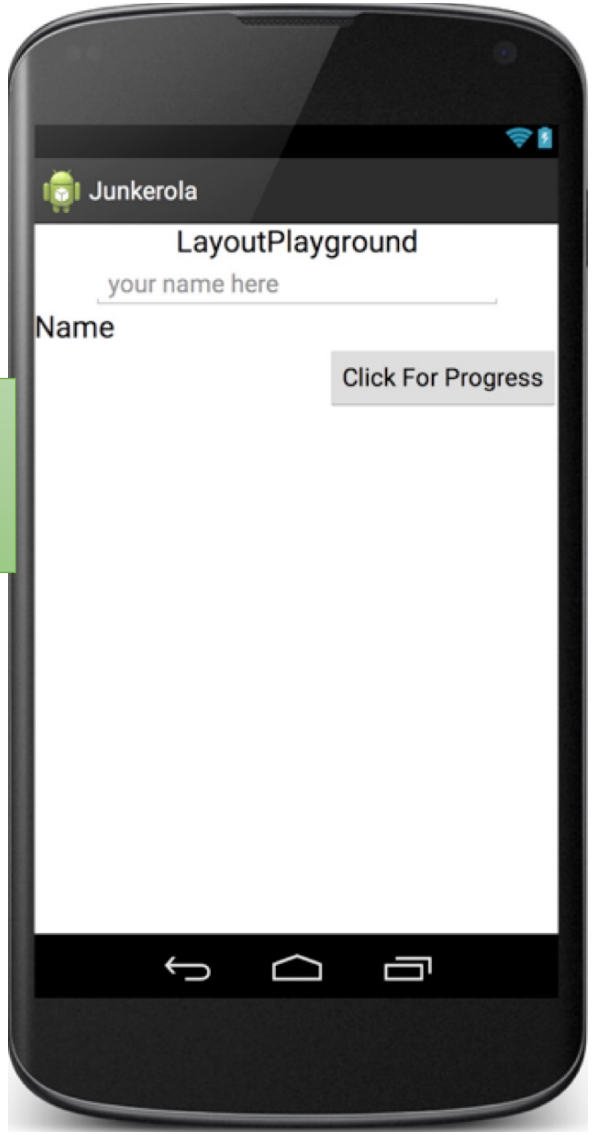
```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="top">


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="LayoutPlayground"
        android:id="@+id/headlinerTV"
        android:layout_gravity="center_horizontal" />


    <EditText
        android:layout_width="300dp"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:id="@+id/nameET"
        android:hint="your name here" />


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Name"
        android:id="@+id/nameTV" />


    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click For Progress"
        android:id="@+id/clickForProgressBTN"
        android:layout_gravity="end" />

</LinearLayout>
```

layout_width & layout_height are w.r.t the parent (probably the screen)

layout_width & layout_height are fixed and sized to the content, respectively

Q: What is the default value if you omit layout_gravity?

Example:



```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent”
    android:layout_height="match_parent"
    android:gravity="top">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView0"
        android:src="@drawable/princess1" />

   <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView1"
        android:layout_gravity="top"
        android:src="@drawable/roundman"
        android:layout_margin="0dp"
        android:background="#ffffef64" />

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView2"
        android:layout_gravity="center_vertical"
        android:src="@drawable/royalperson"
        android:layout_margin="0dp"
        android:background="#ffff390e" />

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView3"
        android:layout_gravity="bottom"
        android:src="@drawable/singer1"
        android:background="#ff24ff13" />
```

Exercise:

```
<ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView4"
        android:layout_gravity="bottom"
        android:src="@drawable/squaregirl"
        android:layout_marginLeft="30dp" />

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imageView5"
        android:layout_gravity="center_vertical"
        android:src="@drawable/womanwaving" />

</LinearLayout>
```

Q: What is drawn? Guess that the screen is about 450dp by 600dp
Images are about 50dp by 100dp

Colors are Alpha, RGB in hex.

Result

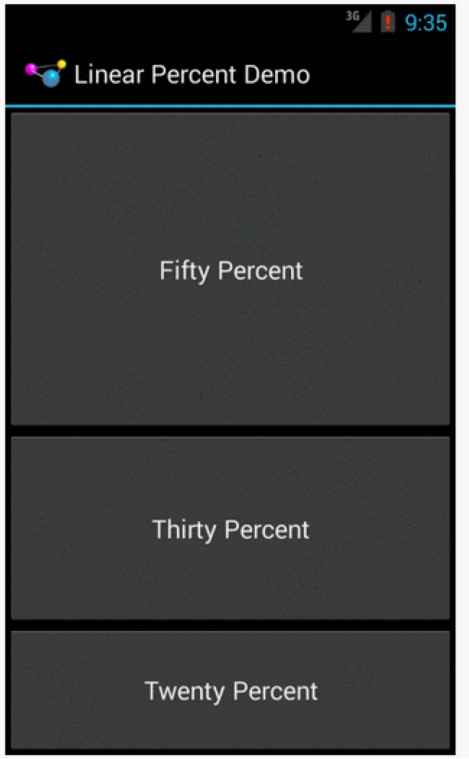
Linear Layouts with fractional sizing

- android:layout_weight is used to assign a weight to each component in the layout
- In a LinearLayout with a *vertical* orientation
    1. Assign android:layout_height=0dp to all components
    2. Assign android:layout_weight values proportional to the % of the layout you wish them to occupy (e.g., 3 components with android:layout_weight="4", "6", and "10") would occupy 20%, 30% and 50% of that linear layout)
- • Why do we haver to bother with step 1? All Views must have both android:layout_width and android:layout_height assigned -- it's the law!
- Horizontal orientation ==> set android:layout_width=0dp

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


  <Button
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="50"
    android:text="@string/fifty_percent"/>


  <Button
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="30"
    android:text="@string/thirty_percent"/>


  <Button
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="20"
    android:text="@string/twenty_percent"/>


</LinearLayout>
```

Excerpt From: Mark L. Murphy. "The Busy Coder's Guide to Android Development Version 6.3." iBooks.

Example



dp or dip are density independent pixels. The goal is to have sizes be similar for devices with different pixel densities.

Gravity

- There are two forms of gravity
  - **android:layout_gravity** – This specifies the position of the component in its parent view group
  - **android:gravity** – This specifies the position of the content of a view.
- e.g., a TextView with **android:layout_gravity=“end"** would mean that in a vertical layout the entire TextView would move to the right, whereas **android:gravity="end"** would have its (content) text move to the right.

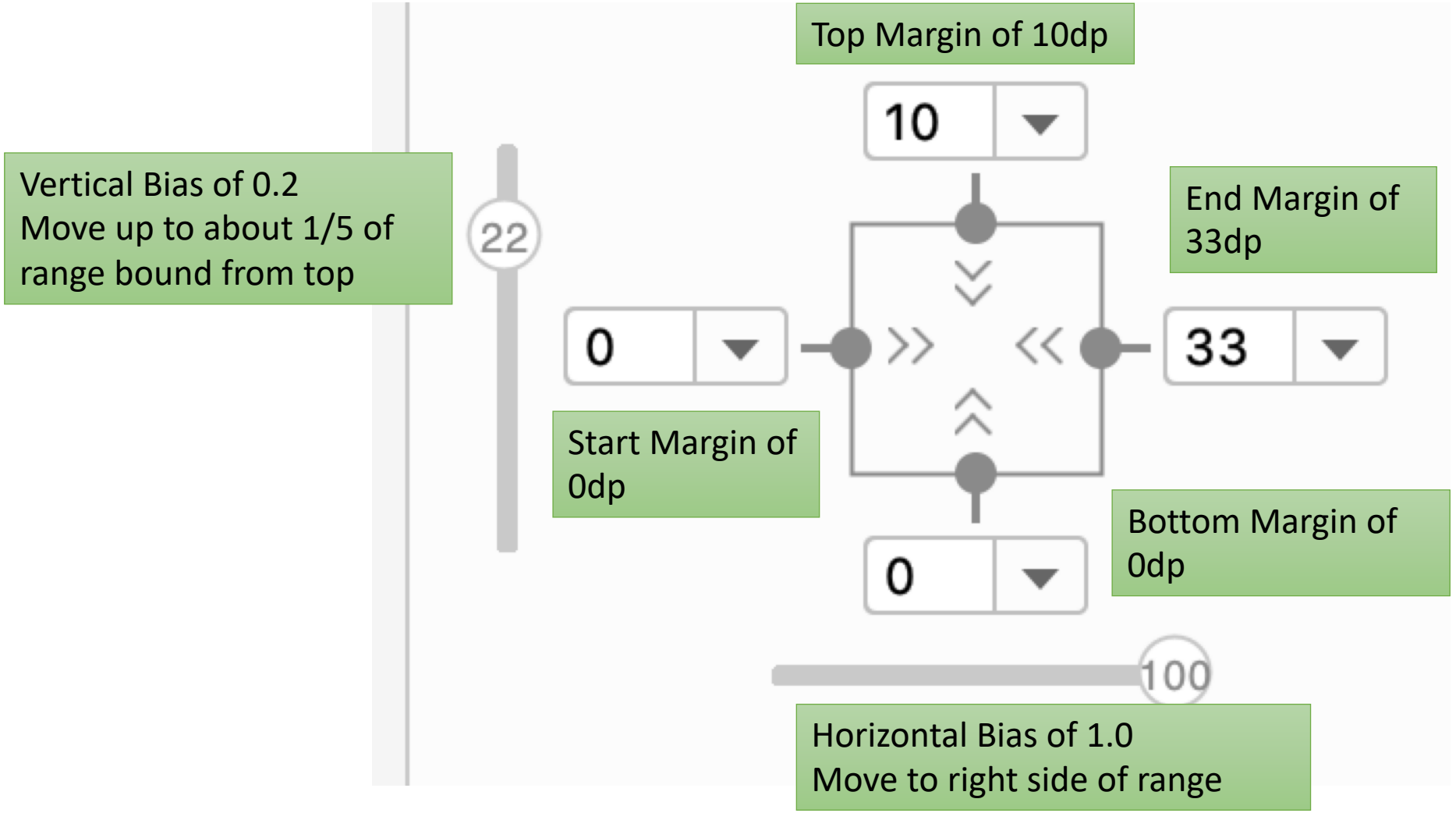
Constraint Layouts

- The position of all children are relative to one another or the parent
- We specify upper and lower bounds (Top/Bottom) for the position of the child.
- We specify left and right bounds (Start/End) for the position of the child.
- Specify the "other" by its resource id or “parent”

Margin/Bias

- Margin we have seen before. It guarantees a certain spacing between the widgets.
- The margin is combined with the constraints to limit the position of the widget. Bias is a value from 0.0 to 1.0 (0% to 100%) and it specifies the position of the widget within its range.
  - horizontal_ bias specifies position between Start and End.
    - 0.0 all the way to the left
    - 0.5 centered
    - 1.0 all the way to the right
  - vertical_bias
- Protip: You can ”fix” the position of the widget by setting margins so the widget has no range to move around in.
- Warning: If the constaints can not be solved, the widget will be positioned off screen in the upper left corner.

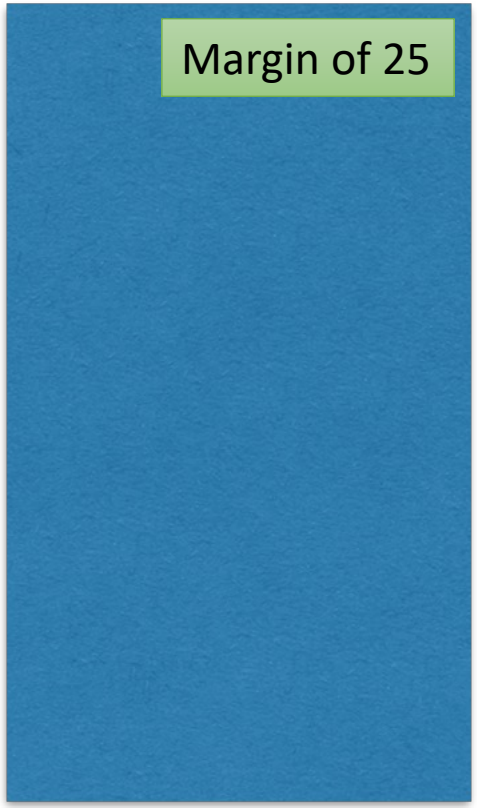
Inspector Control Margin/Bias



Constrain to Parent

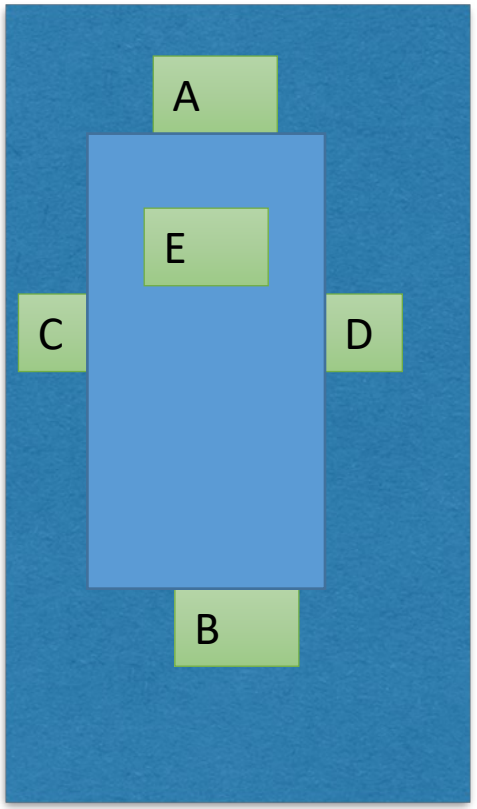
- app:layout_constraintTop_toTopOf="parent"
  - widget top will be constrained to top of parent
- app:layout_constraintBottom_toBottomOf="parent"
  - widget bottom will be constrained to parent bottom
- app:layout_constraintStart_toStartOf="parent"
  - widget start will be constrained to with parent start
- app:layout_constraintEnd_toEndOf="parent"
  - widget end will be constrained to with parent end

- app::layout_constraintHorizontal_bias="1.0"
  - All the way to the right to what it is constrained with (edge of parent + margin)
- app:layout_constraintVertical_bias="0.0"
  - All the way to the top of what it is constrained with



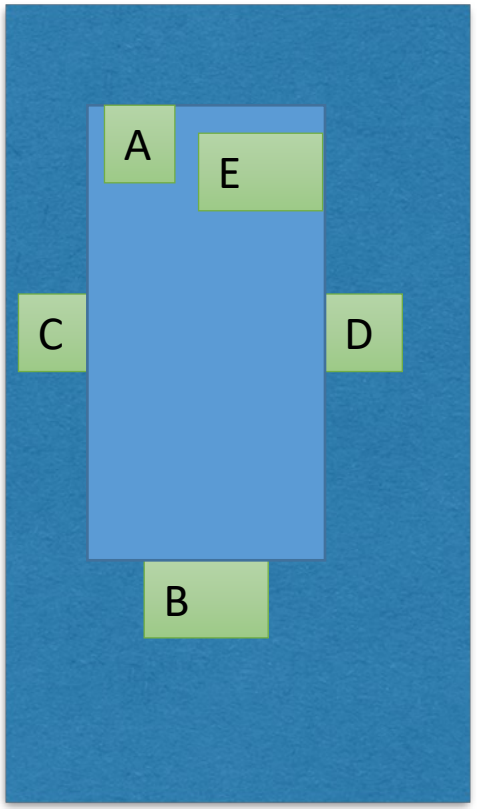
Constrain to Other

- app:layout_constraintTop_toBottomOf="@+id/widgetA
  - "widget top will be constrained to bottom of the named widget (E is below A)
- app:layout_constraintBottom_toTopOf="@+id/widgetB
  - "widget bottom will be constrained to top of the named widget (E is above B)
- app:layout_constraintStart_toEndOf="@+id/widgetC
  - "widget start will be constrained to end of the named widget (E is right of C)
- app:layout_constraintEnd_toStartOf="@+id/widgetD
  - "widget end will be constrained to start of the named widget (E is left of D)
- In all cases specify the id of another widget
- Ids are specified with @+id/widgetName the 1st time they appear in xml file; in subsequent appearances, just write @id/widgetName
- Margin and Bias will control where E is inside the light blue range.



Constrain to Other

- app:layout_constraintTop_toTopOf="@+id/widgetA
  - "widget top will be constrained to top of the named widget (E is aligns with A)
- app:layout_constraintBottom_toTopOf="@+id/widgetB
  - "widget bottom will be constrained to top of the named widget (E is above B)
- app:layout_constraintStart_toEndOf="@+id/widgetC
  - "widget start will be constrained to end of the named widget (E is right of C)
- app:layout_constraintEnd_toStartOf="@+id/widgetD
  - "widget end will be constrained to start of the named widget (E is left of D)
- Margin and Bias again controls where E is inside the light blue range, but we might overlap with A this time.



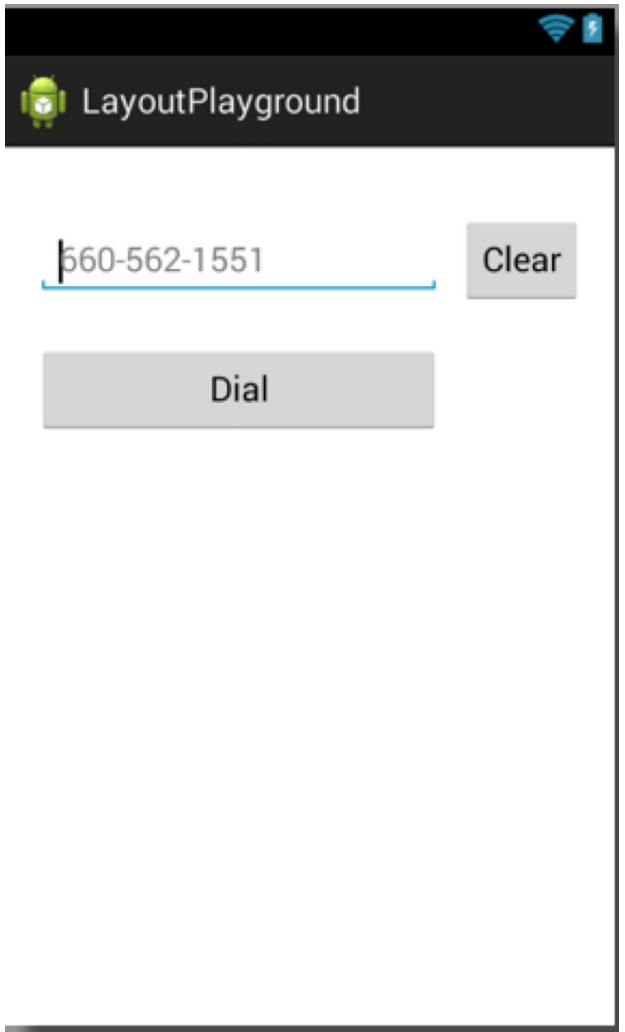
Align Text

- app:layout_constraintBaseline_toBaselineOf
  - align this widget's text base line with the other's widgets baseline
- Useful when you have widgets using text of a different size.

Example (Legacy)

```
<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" >

    <EditText
        android:id="@+id/phoneET"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginTop="26dp"
        android:ems="10"
        android:hint="660-562-1551"
        android:inputType="phone" >
        <requestFocus />
    </EditText>

    <Button
        android:id="@+id/clearBTN"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/phoneET"
        android:layout_alignBottom="@+id/phoneET"
        android:layout_alignParentRight="true"
        android:text="Clear" />


    <Button
        android:id="@+id/dialBTN"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/phoneET"
        android:layout_alignRight="@+id/phoneET"
        android:layout_below="@+id/clearBTN"
        android:layout_marginTop="25dp"
        android:text="Dial" />
</RelativeLayout>
```

EMS – Width of EditText using width of character m in the current font.



Grid Layouts

- GridLayouts are useful when you need to arrange children in a rectangular grid
  - Children allowed to span multiple cells
  - Shape of children must be rectangular

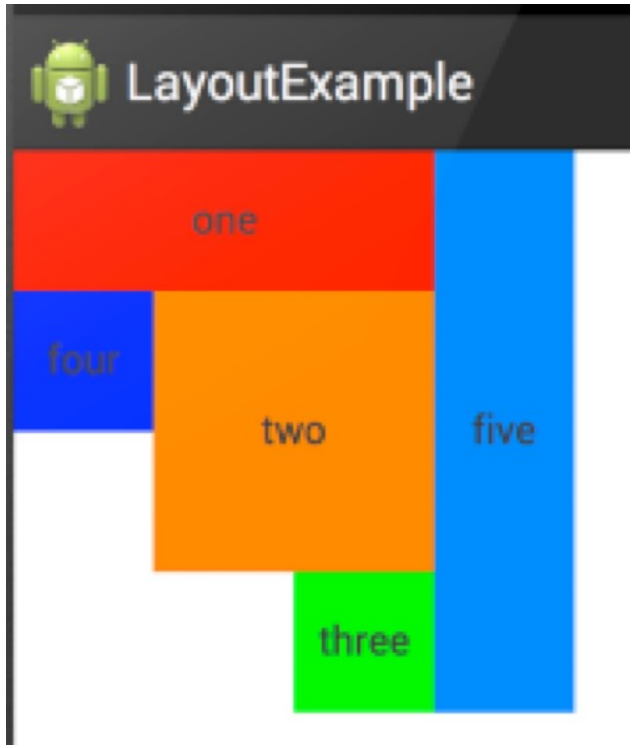
- Each child is placed in a particular row and column, with indexing starting at 0
- The space for each row and col is determined by the size of its contents.

GridLayout Attributes

Attribute Name (all begin with `android:`)	Applies to	Description	Value
`columnCount`	`GridLayout`	Defines a fixed number of columns for the grid.	A whole number; for example, `4`.
`rowCount`	`GridLayout`	Defines a fixed number of rows for the grid.	A whole number; for example, `3`.
`orientation`	`GridLayout`	When a row or column value is not specified on a child, this is used to determine whether the next child is down a row or over a column.	Can be vertical (down a row) or horizontal (over a column).
`layout_column`	Child `View` of `GridLayout`	Index of the column this child `View` should be displayed in (zero based).	Integer or integer resource; for example, `1`.
`layout_columnSpan`	Child `View` of `GridLayout`	Number of columns this child `View` should span across.	Integer or integer resource greater than or equal to 1; for example, `3`.
`layout_row`	Child `View` of `GridLayout`	Index of the row this child `View` should be displayed in (zero based).	Integer or integer resource; for example, `1`.
`layout_rowSpan`	Child `View` of `GridLayout`	Number of rows this child `View` should span down.	Integer or integer resource greater than or equal to 1; for example, `3`.
`layout_gravity`	Child `View` of `GridLayout`	Specifies the "direction" in which the `View` should be placed within the grid cells it will occupy.	One or more constants separated by "|". The constants available are `baseline`, `top`, `bottom`, `left`, `right`, `center_vertical`, `fill_vertical`, `center_horizontal`, `fill_horizontal`, `center`, `fill`, `clip_vertical`, `clip_horizontal`, `start`, and `end`. Defaults to `LEFT|BASELINE`.

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout
xmlns:android="http://schemas.android.com
/apk/res/android"
    android:id="@+id/gridLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="4"
    android:rowCount="4" >
    <TextView
        android:layout_width="150dp"
        android:layout_height="50dp"
        app:layout_column="0"
        app:layout_columnSpan="3"
        app:layout_row="0"
        android:background="#ff0000"
        android:gravity="center"
        android:text="one" />
    <TextView
        android:layout_width="100dp"
        android:layout_height="100dp"
        app:layout_column="1"
        app:layout_columnSpan="2"
        app:layout_row="1"
        app:layout_rowSpan="2"
        android:background="#ff7700"
        android:gravity="center"
        android:text="two" />
    <TextView
        android:layout_width="50dp"
        android:layout_height="50dp"
        app:layout_column="2"
        app:layout_row="3"
        android:background="#00ff00"
        android:gravity="center"
        android:text="three" />
    <TextView
        android:layout_width="50dp"
        android:layout_height="50dp"
        app:layout_column="0"
        app:layout_row="1"
        android:background="#0000ff"
        android:gravity="center"
        android:text="four" />
    <TextView
        android:layout_width="50dp"
        android:layout_height="200dp"
        app:layout_column="3"
        app:layout_row="0"
        app:layout_rowSpan="4"
        android:background="#0077ff"
        android:gravity="center"
        android:text="five" />
</GridLayout>
```

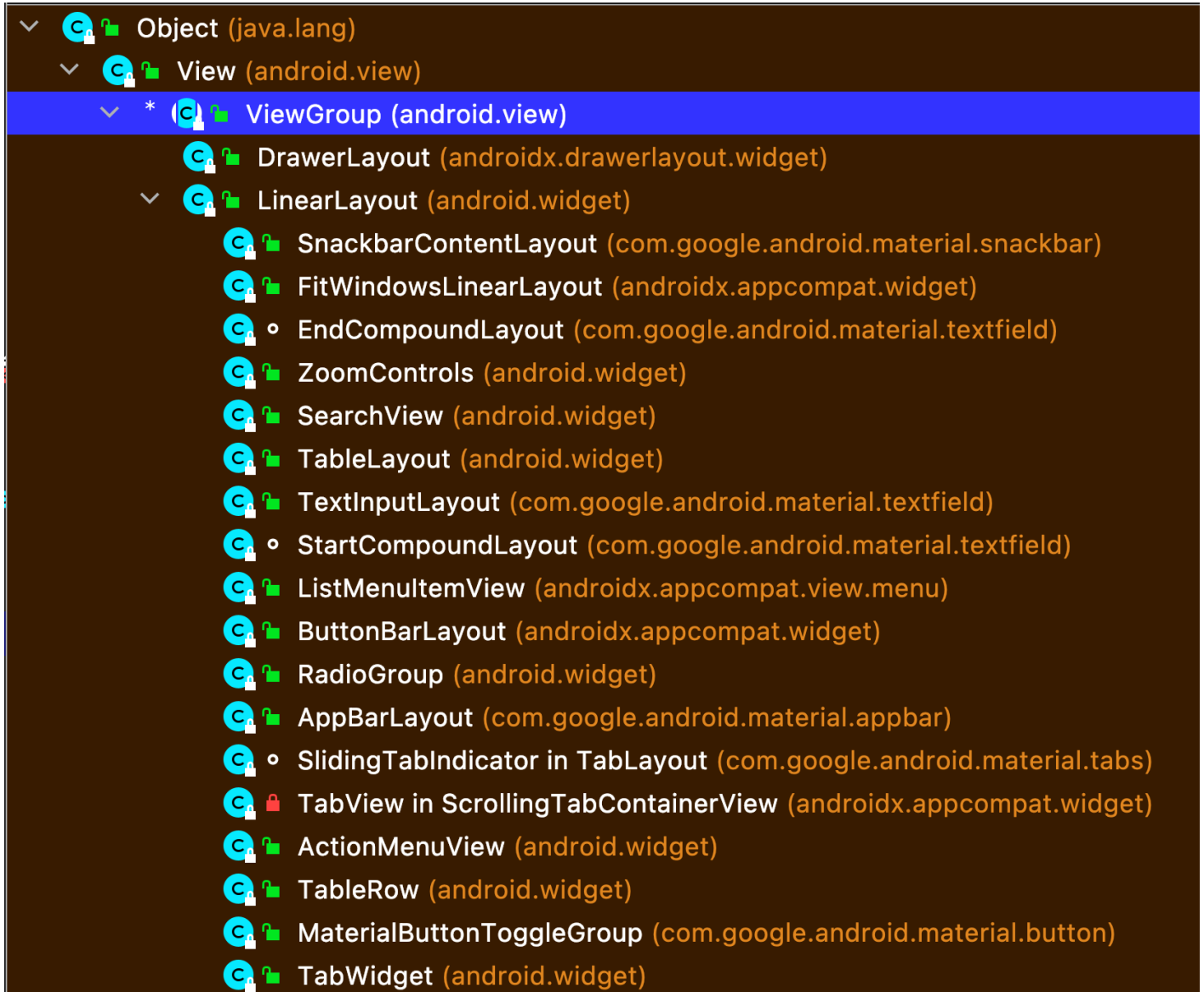
Grid Example



Colors are RGB in hex.

The Hierarchy (part)

- A View is the base class for anything visible on the screen
- A ViewGroup can contain other Views
- Forms the base class for Layouts
- Defines a nested class, ViewGroup.LayoutParams
- Androidx is a newer set of redesigned widgets.



Programatic Example

```
public class MainActivity extends Activity {

   final static double MAX_PRICE = 100.00;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      // setContentView(R.layout.activity_main);
      LinearLayout linear = new LinearLayout(this);
      linear.setLayoutParams(new LayoutParams(
         LayoutParams.MATCH_PARENT,LayoutParams.MATCH_PARENT));

      linear.setOrientation(LinearLayout.VERTICAL);
      linear.setBackgroundColor(Color.LTGRAY);
      Button btn = new Button(this);
      btn.setText("Click Me");

      LinearLayout.LayoutParams params = new
         LinearLayout.LayoutParams(
            LinearLayout.LayoutParams.WRAP_CONTENT,
            LinearLayout.LayoutParams.WRAP_CONTENT);

      params.gravity = Gravity.CENTER_HORIZONTAL;
      btn.setGravity(Gravity.RIGHT); //sets text alignment
      final EditText et = new EditText(this);
      et.setHint("name your price");
      btn.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            et.setText(String.format("$%.2f", Math.random()*MAX_PRICE));
         }
      });

   linear.addView(btn, params);
   linear.addView(et, params);
   setContentView(linear);
}
```

Color constant from early Java.

Is this going to be on the exam?

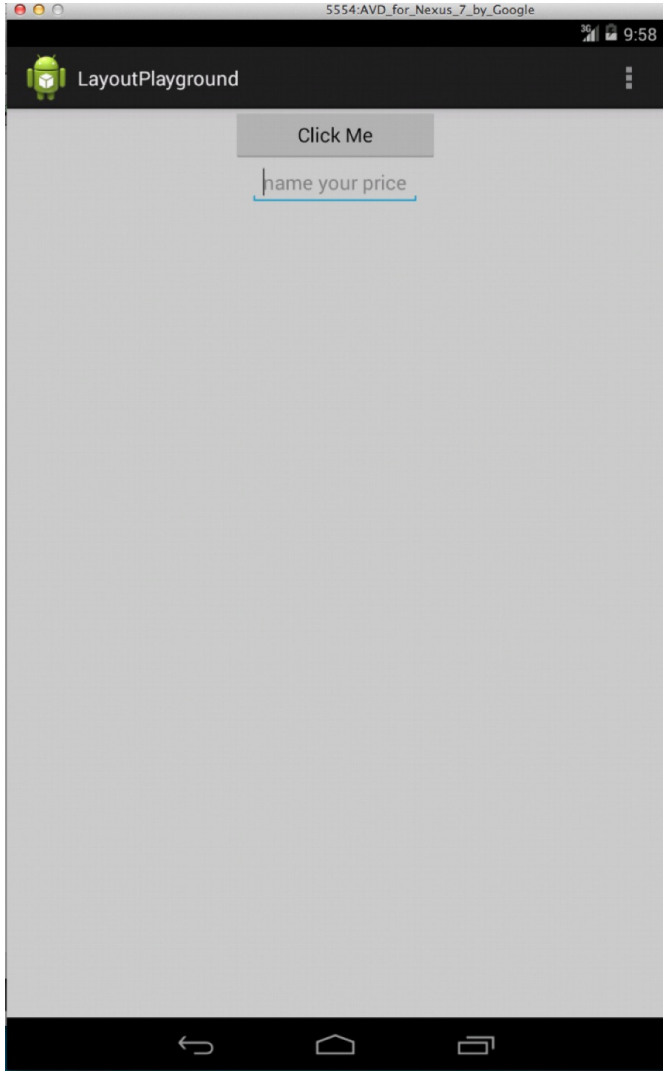
The Result

- That was a lot of work.

You can create an app without ever specifying a layout in XML.

The advantage of working with the resource file is that you have a graphical interface where you can adjust parameters/code and see the effect immediately.

There will be situations though, where we will need to change the View on screen which will require a programmatic approach.



Questions

1. What is the purpose of a layout file?
2. What is the relationship between XML element names and Android classes? Between XML attributes and Android methods?
3. What 2 layout attributes must be specified?
4. Explain the difference between match_parent and wrap_content. Where are they used?
5. What is the difference between @id and @+id?
6. Write a complete onCreate() method to load a layout stored in the file activity_waterloo.xml
7. Distinguish between a View, ViewGroup and Layout
8. Draw a picture showing a View's content, border, padding, and margin.
9. A View has content that is 50x30, padding of 5x5x5x5, and a margin of 10x10x10x10. What is the size of the View?
10. What is the difference between a LinearLayout and a ConstraintLayout? Write the attribute so that a LinearLayout lays its elements out horizontally.

Questions 2

11. What does it mean to say that an XML element has android:layout_gravity=“end" ?
12. Which widget has an android:src attribute?
13. What is the difference between android:gravity="start" & android_layout_gravity="start"
14. Define the following terms: baseline, descender height and ascender height
15. Name 2 other layouts, and where they might be used
16. Arrange LinearLayout, Object, ViewGroup and View into the correct hierarchy

Questions 3

17. For each attribute, identify which layout(s) it applies to, and its significance
    1. android:layout_constraintStart_toStartOf
    2. android:layout_width
    3. android:id
    4. android:layout_gravity
    5. android:layout_constraintStart_toEndOf
    6. android:layout_constraintBottom_toTopOf

Resources

- Andoid Developer Layouts
  - Linear Layouts
  - Relative Layouts
  - Training on Constraint Layouts
  - Grid Layouts
- http://mainerrors.blogspot.com/2011/02/programmatically-creating-layout-part-1.html