CS371m - Mobile Computing

User Interface Basics

Clicker Question

- Have you ever implemented a Graphical User Interface (GUI) as part of a program?

A. Yes, in another class.

B. Yes, at a job or internship.

C. Yes, on my own.

D. No

Android UIs

- An Android Activity is a ***single, focused thing the user can do.***
- Has code in a class
- Has a user interface
- One of the four main components Android uses to interact with our code / program / app
- **Demo apps and point out activities**

Android UIs

- An Activity has an associated layouts for their UI
  - Can be several, typically just one that might change slightly
- Layouts are declared in XML files
- Layouts consist of various *Views*
  - View is an Android class that represents a ***rectangular area*** on the screen and is responsible for ***drawing*** and ***event handling***.
  - many, many, many sublcases

VIEWGROUPS - TOP LEVEL CONTAINERS FOR USER INTERFACES

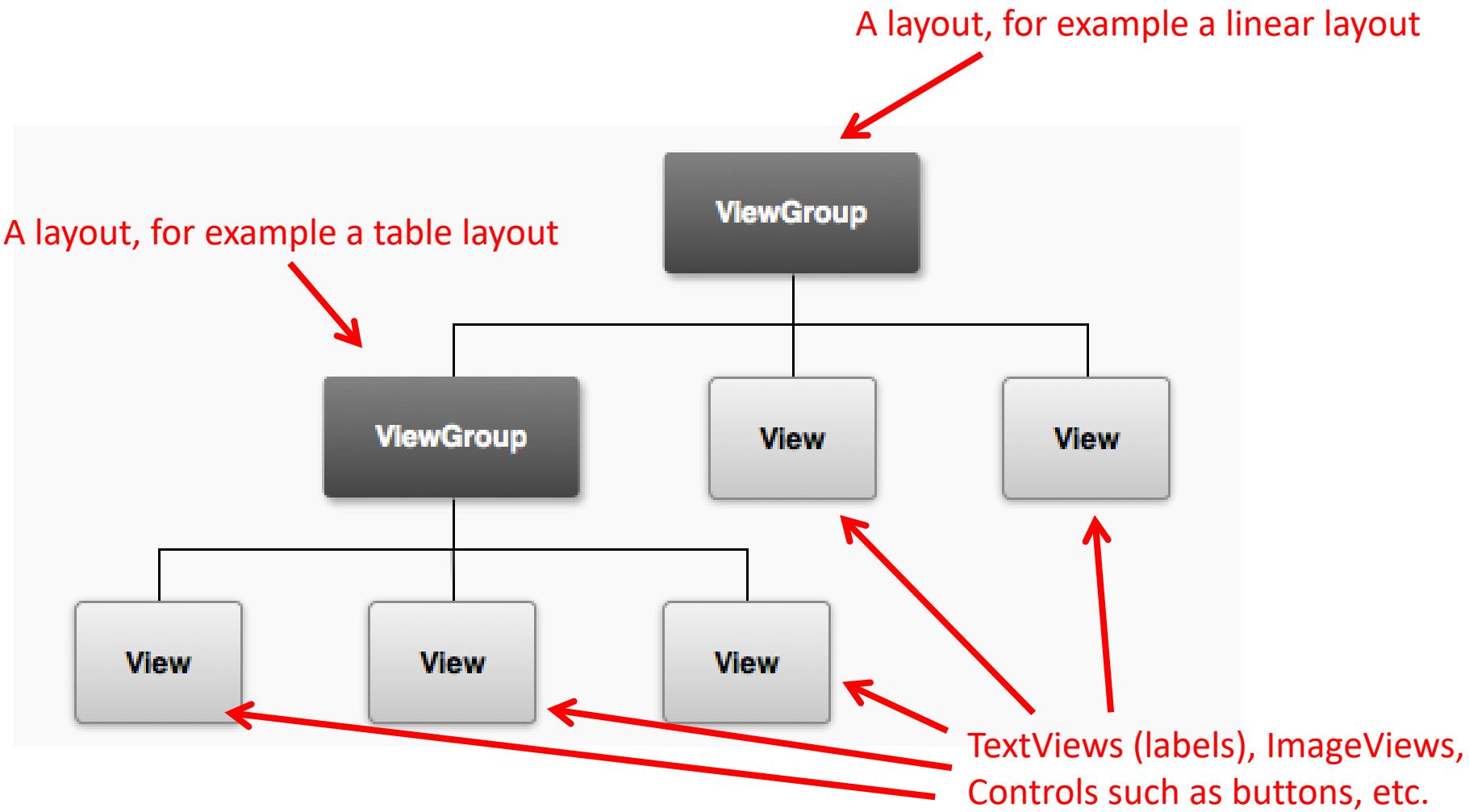
ViewGroups - Layouts

- Layouts are subclasses of ViewGroup
  - Which is a subclass of View.
- Still a view but doesn't actually draw anything.
- serve as a containers for other views
  - similar to Java layout managers
  - you can nest ViewGroups
- options on how sub views (and view groups) are arranged
- FrameLayout, LinearLayout, TableLayout, GridLayout, RelativeLayout, ListView, GridView, ScrollView, DrawerLayout, ViewPager, AbsoluteLayout, RecyclerView, and more!
- **Demo developer options, show layout bounds**

ViewGroups - Containers

- View are used to organize multiple widgets into a structure
- Similar to layout managers in Java
- Children can be UI widgets or other containers
- ViewGroups have a set of rules governing how it lays out its children in the screen space the container occupies

Containers (ViewGroups) and Widgets (Views)



XML UI Configuration

- Layouts can contain UI elements (built in Android and programmer created)
- res/layout
- "Design by Declaration"
- why?
- tools to parse XML to display result in a graphical way
  - build drag and drop editors

UI Via XML

- Each Screen in your app will likely have an xml layout file
- describes the container and widgets on the screen / UI
- Edit xml or use drag and drop editor
- alter container and layout attributes for the set up you want
- we will then access and manipulate the container and widgets in our Java code associated with the UI / screen.

FrameLayout

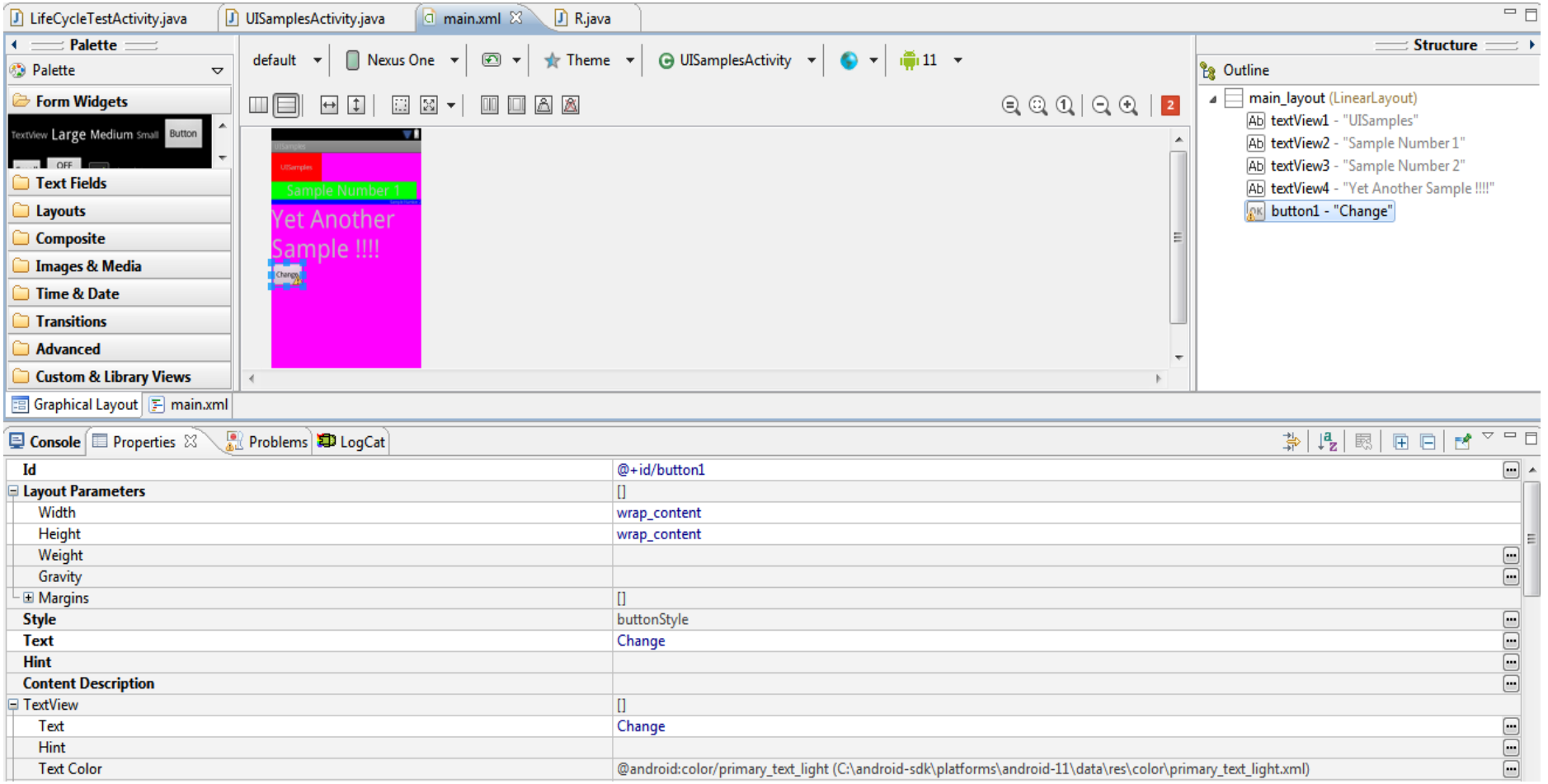
- FrameLayout
  - simplest type of layout object
  - fill with a single object (such as a picture) that can be switched in and out
  - child elements pinned to top left corner of screen and cannot be move
  - adding a new element / child draws over the last one

LinearLayout

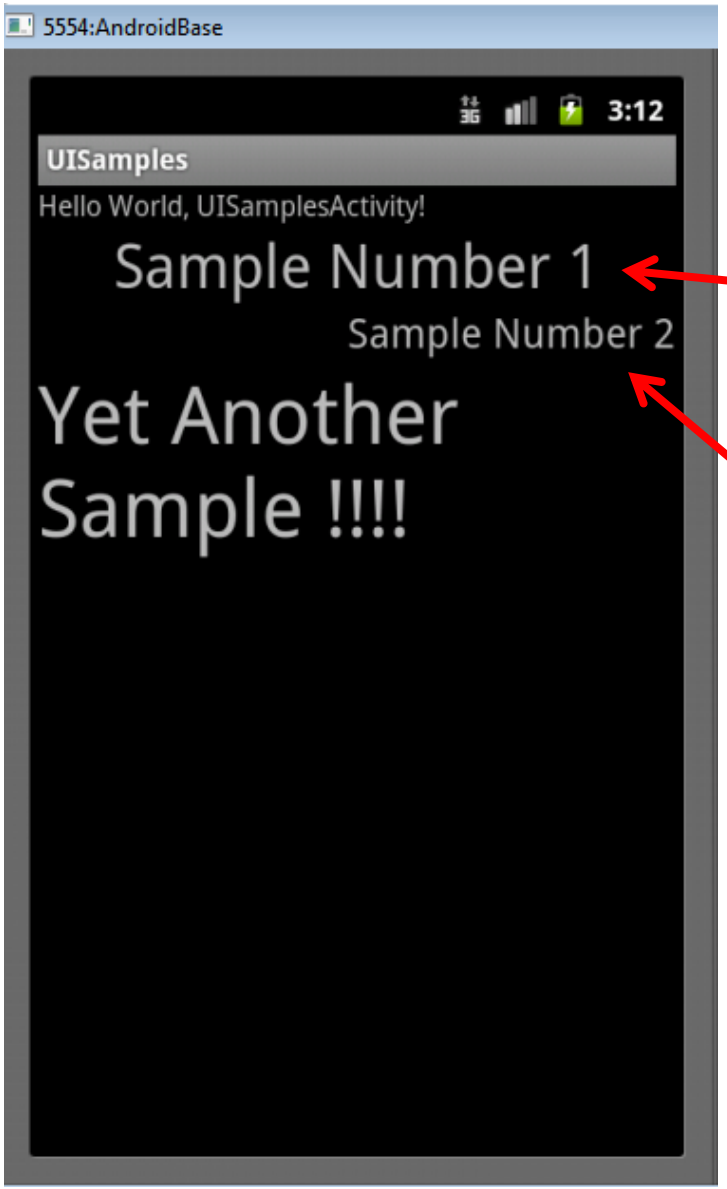
- aligns child elements (such as buttons, edit text boxes, pictures, etc.) in a single direction
- orientation attribute defines direction:
  - android:orientation="*vertical*"
  - attribute of View

Modifying Attributes

- in xml, programmatically, and visual editor



Gravity Attribute



- Child element's gravity attribute
  - where to position in the outer container

Weight

- layout_weight attribute
  - "importance" of a view
  - default = 0
  - if set > 0 takes up more of parent space

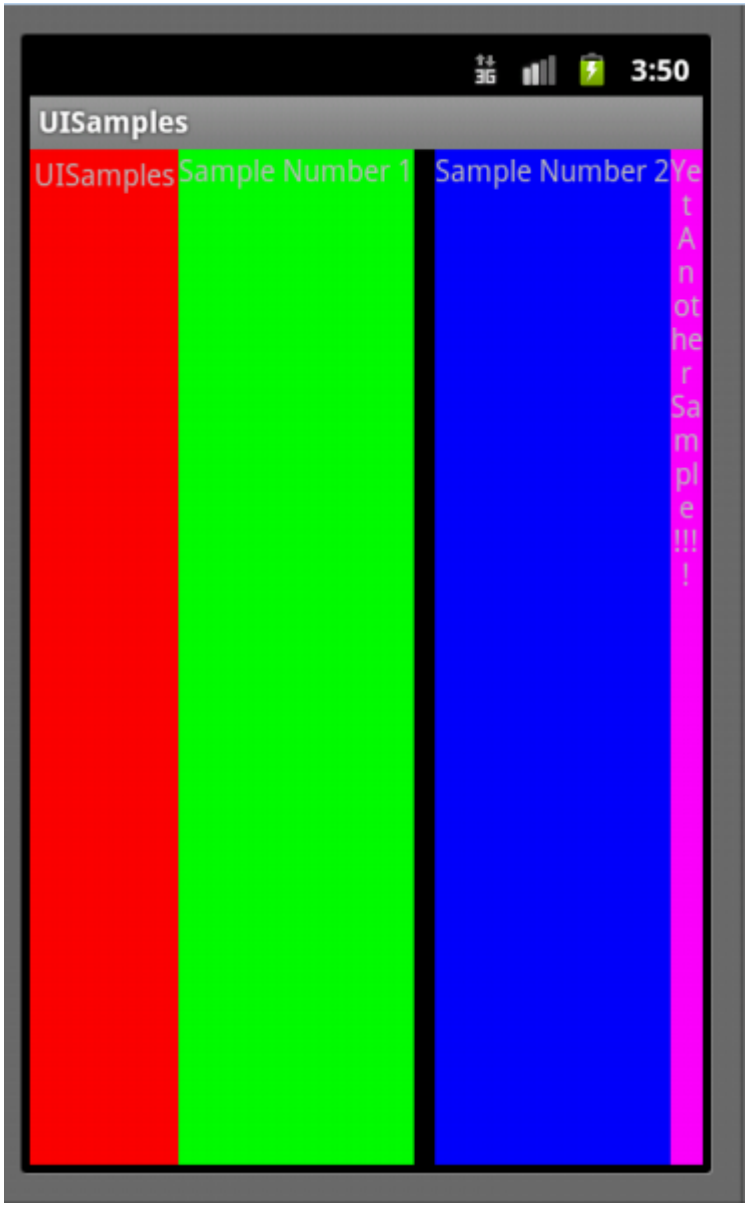
Another Weight Example

button and bottom edit text weight of 2

button weight 1 and bottom edit text weight of 2

LinearLayout - Horizontal Orientation

- padding
- background color
- margins

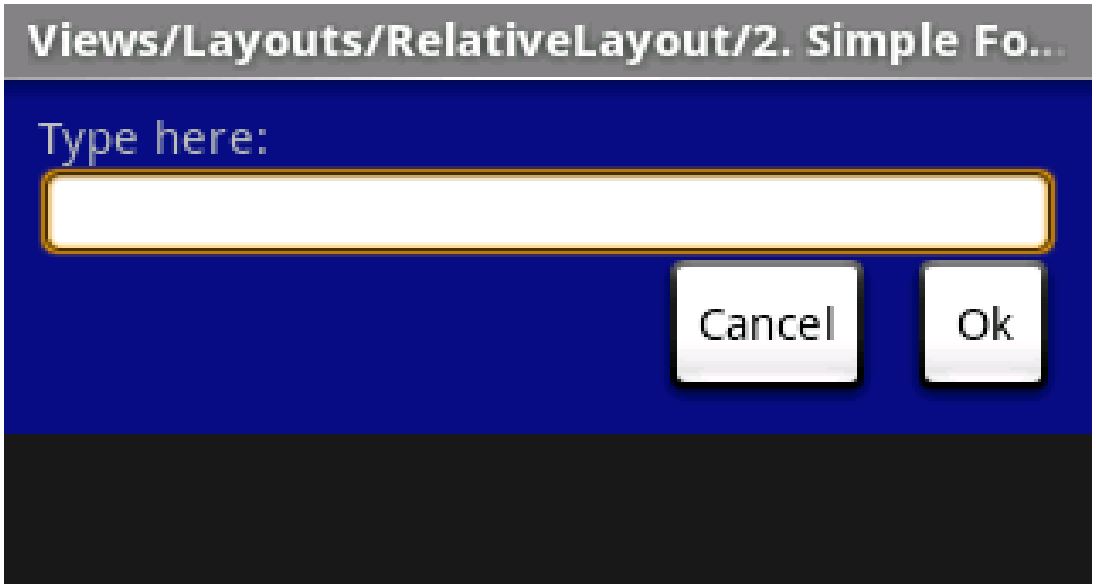


TableLayout

- rows and columns
- rows normally TableRows (subclass of LinearLayout)
- TableRows contain other elements such as buttons, text, etc.

RelativeLayout

- children specify position relative to parent or to each other (specified by ID)
- First element listed is placed in "center"
- other elements placed based on position to other elements



RelativeLayout XML

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

    <TextView android:id="@+id/label"
              android:layout_width="fill_parent"
              android:layout_height="wrap_content"
              android:text="Type here:" />

    <EditText android:id="@+id/entry"
              android:layout_width="fill_parent"
              android:layout_height="wrap_content"
              android:background="@android:drawable/editbox_background"
              android:layout_below="@id/label" />
```

RelativeLayout XML

```
    <Button android:id="@+id/ok"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@id/entry"
            android:layout_alignParentRight="true"
            android:layout_marginLeft="10px"
            android:text="OK" />

    <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_toLeftOf="@id/ok"
            android:layout_alignTop="@id/ok"
            android:text="Cancel" />
</RelativeLayout>
```

GridLayout

- added in Android 4.0
- child views / controls can span multiple rows and columns
  - different than TableLayout
- child views specify row and column they are in or what rows and columns they span

Container Control Classes

- Layouts shown are useful for positioning UI elements
  - the layouts themselves are not interactive although the child Views may be
- Other available layouts add a level of interactivity between the user and the child Views
- ListView, GridView, GalleryView
- Tabs with TabHost, TabControl
- ScrollView, HorizontalScrollView

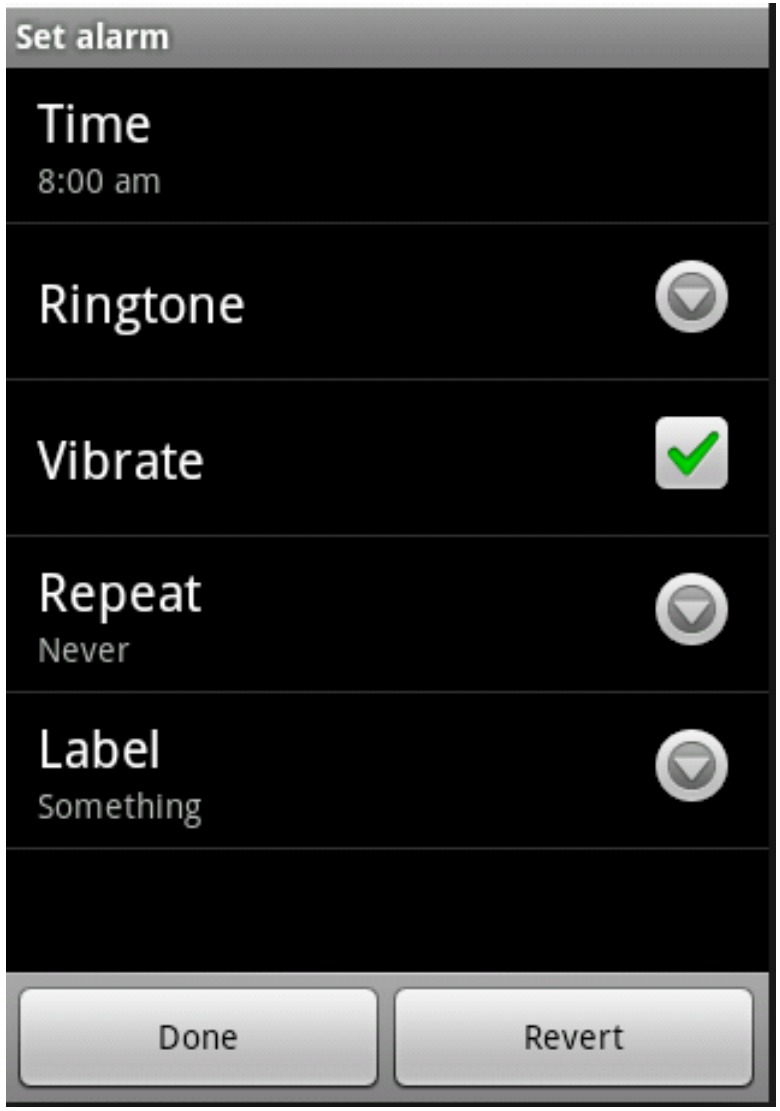
USER INTERFACE ELEMENTS
WIDGETS

UI Programming with Widgets

- Widgets are an element in a Graphical User Interface (GUI)
  - not to be confused with app widgets placed on the home screen, mini version of app
- Widgets are building blocks
- User interacts with a given widget
- **Often** use prebuilt widgets
  - Advanced developers create their own (Chris Renke, Square)

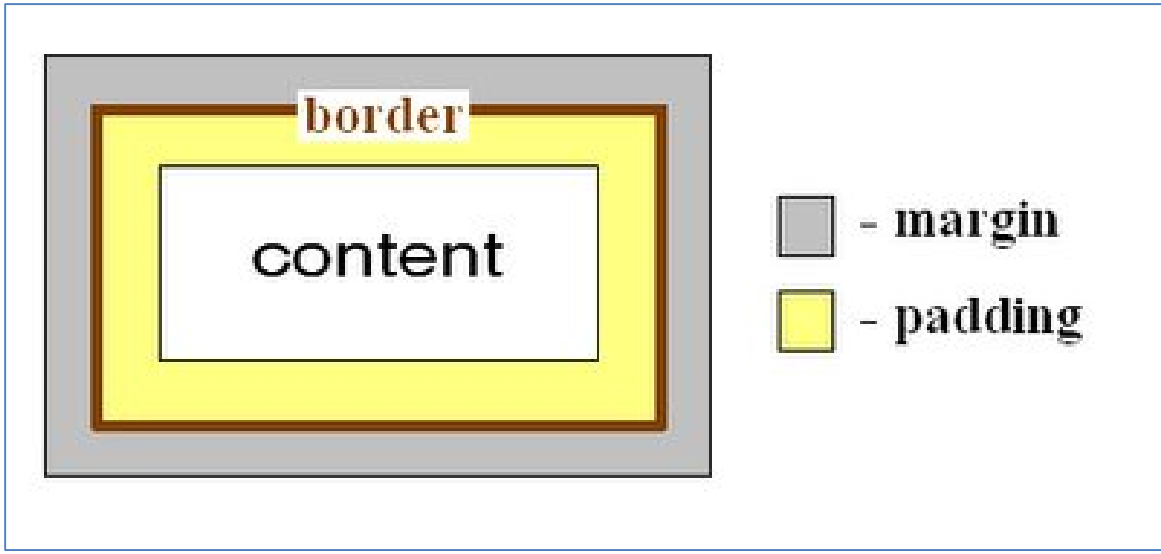
Widgets

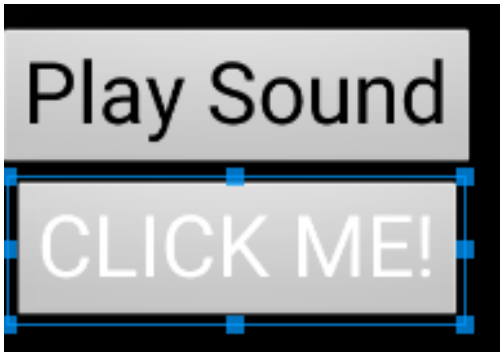
- Including:
- Text Views
- EditTexts
- Buttons
- Check Boxes
- Spinners (drop down menus)
- and many, many more



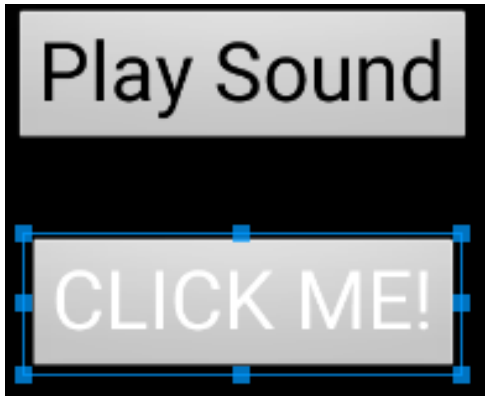
Widget Attributes

- Size
  - layout width
  - layout height
- Margin
- Padding





No specified margin
or padding



Top Margin of 30dp
(density independent pixels)

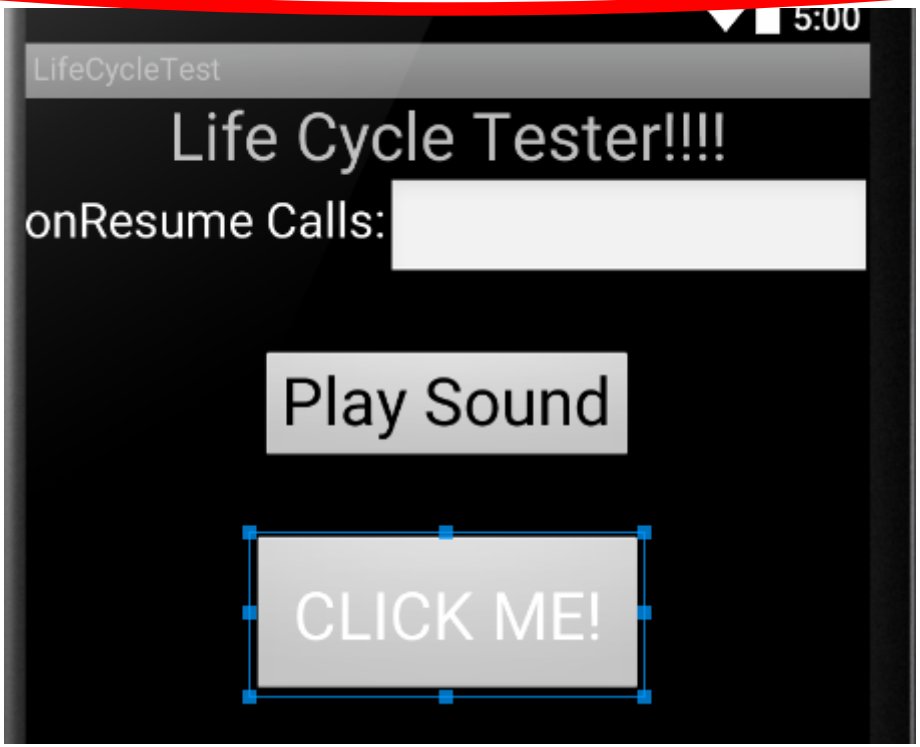
Top Margin of 30dp,
padding of 20dp

Size

- Three options:
- Specified (hard coded) size in dp, density independent pixels
- wrap_content
  - widget is just big enough to show content inside the widget (text, icon)
- match_parent
  - match my parent's size
  - widgets stored in a *container or ViewGroup*

Size - Wrap Content

```
<Button
    android:id="@+id/clickForActivityButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

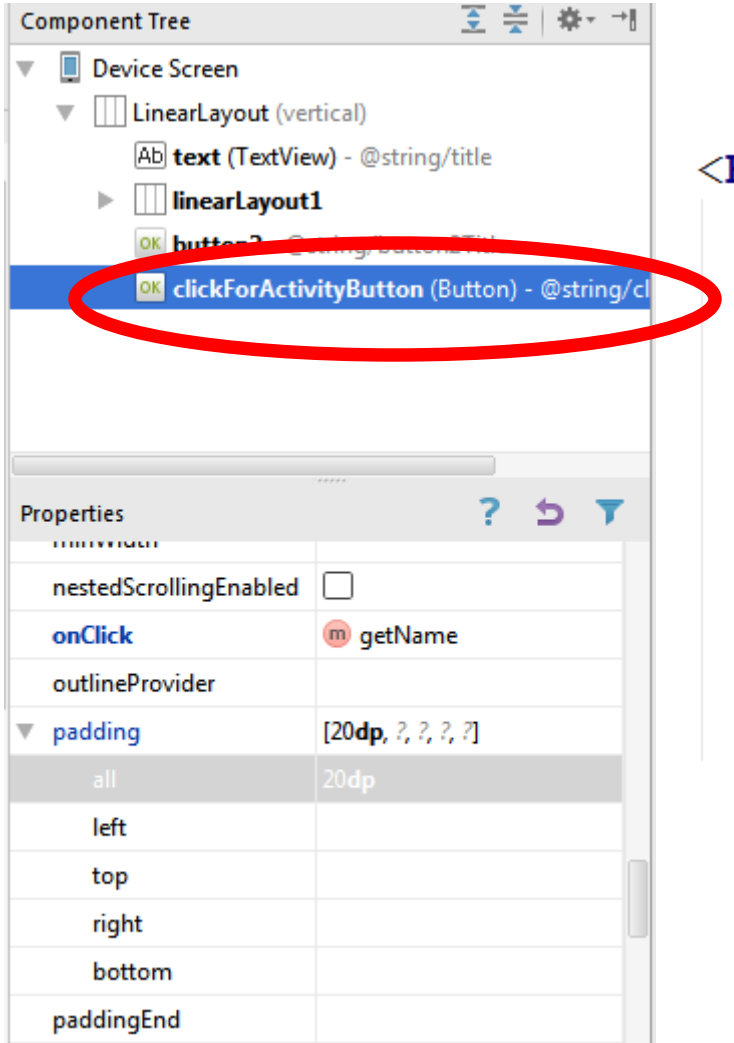


Size - Match Parent

```
<Button
    android:id="@+id/clickForActivityButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

Widgets and Android Studio

- GUI for GUI design and XML




```
<Button
    android:id="@+id/clickForActivityButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_marginTop="30dp"
    android:onClick="getName"
    android:padding="20dp"
    android:text="@string/clickForActivityB
    android:textColor="#FFF"
    android:textSize="30sp" />
```

Attributes

- *android:padding="20dp"* appears in the xml file for the button and sets the given attribute to the specified value
- see the view class or appropriate sub class for attributes
  - a lot of attributes
- http://tinyurl.com/y8jj5eo
- attributes can be set in the xml and most can changed programmatically

Attributes

XML Attributes		
Attribute Name	*Related Method*	*Description*
android:baselineAligned	setBaselineAligned(boolean)	When set to false, prevents the layout from aligning its children's baselines.
android:baselineAlignedChildIndex	setBaselineAlignedChildIndex(int)	When a linear layout is part of another layout that is baseline aligned, it can specify which of its children to baseline align to (that is, which child TextView).
android:divider	setDividerDrawable(Drawable)	Drawable to use as a vertical divider between buttons.
android:gravity	setGravity(int)	Specifies how to place the content of an object, both on the x- and y-axis, within the object itself.
android:measureWithLargestChild	setMeasureWithLargestChildEnabled(boolean)	When set to true, all children with a weight will be considered having the minimum size of the largest child.
android:orientation	setOrientation(int)	Should the layout be a column or a row? Use "horizontal" for a row, "vertical" for a column.
android:weightSum		Defines the maximum weight sum.

Inherited XML Attributes [Expand]

▼ From class android.view.ViewGroup

Attribute Name	*Related Method*	*Description*
android:addStatesFromChildren		Sets whether this ViewGroup's drawable states also include its children's drawable states.
android:alwaysDrawnWithCache		Defines whether the ViewGroup should always draw its children using their drawing cache or not.
android:animateLayoutChanges	setLayoutTransition(LayoutTransition)	Defines whether changes in layout (caused by adding and removing items) should cause a LayoutTransition to run.
android:animationCache		Defines whether layout animations should create a drawing cache for their children.
android:clipChildren	setClipChildren(boolean)	Defines whether a child is limited to draw inside of its bounds or not.
android:clipToPadding	setClipToPadding(boolean)	Defines whether the ViewGroup will clip its drawing surface so as to exclude the padding area.
android:descendantFocusability		Defines the relationship between the ViewGroup and its descendants when looking for a View to take focus.
android:layoutAnimation		Defines the layout animation to use the first time the ViewGroup is laid out.

```
<Button
    android:id="@+id/clickForActivityButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_marginTop="30dp"
    android:onClick="getName"
    android:padding="20dp"
    android:text="@string/clickForActivityButtonTitle"
    android:textColor="#FFF"
    android:textSize="30sp" />
```

in layout xml file

```
private void changeButtonPadding() {
    Button b = (Button) findViewById(R.id.clickForActivityButton);
    b.setPadding(20, 15, 20, 15);
}
```

Programmatically in Activity (Java code)

in program

Clicker

- What is the purpose of the xml files in the res/layout directory in an Android project?

A. define all the Java classes in the project

B. define user interfaces

C. localize String resources

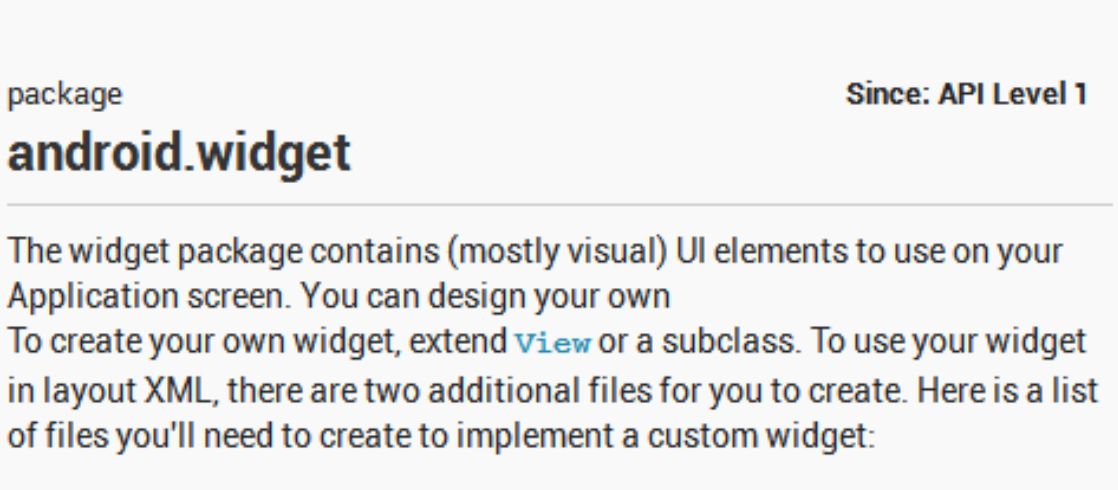
D. store graphic image resources such as jpeg and png files

E. list the permissions the app requests

TYPES OF WIDGETS

Android Controls

- android.widget package
- Not to be confused with application widgets, mini versions of applications
- Still subclasses of View
- interactive components of the UI
  - layouts are the containers



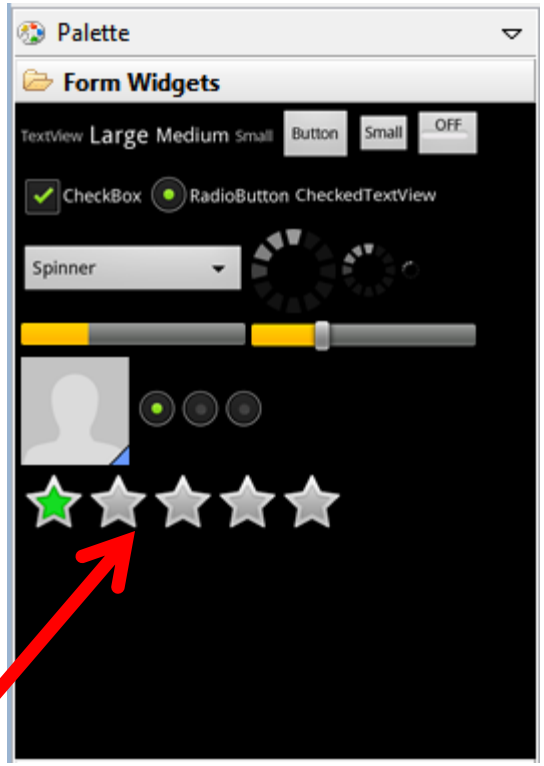
package **Since: API Level 1**

android.widget

The widget package contains (mostly visual) UI elements to use on your Application screen. You can design your own
To create your own widget, extend `View` or a subclass. To use your widget in layout XML, there are two additional files for you to create. Here is a list of files you'll need to create to implement a custom widget:

Adding Controls

- Widgets can be added to the XML layout or at run time
- Add component in visual editor and XML code automatically generated
- tweak XML code as desired




```
<RatingBar
    android:id="@+id/ratingBar1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Common Controls - TextView

- a simple label
- display information, not for interaction
- common attributes: width, height, padding, visibility, text size, text color, background color
  - units for width / height: px (pixels), dp or dip (density-independent pixels 160 dpi base), sp (scaled pixels based on preferred font size), in (inches), mm (millimeters)
  - recommended units: sp for font sizes and dp for everything else
  - http://developer.android.com/guide/topics/resources/more-resources.html#Dimension

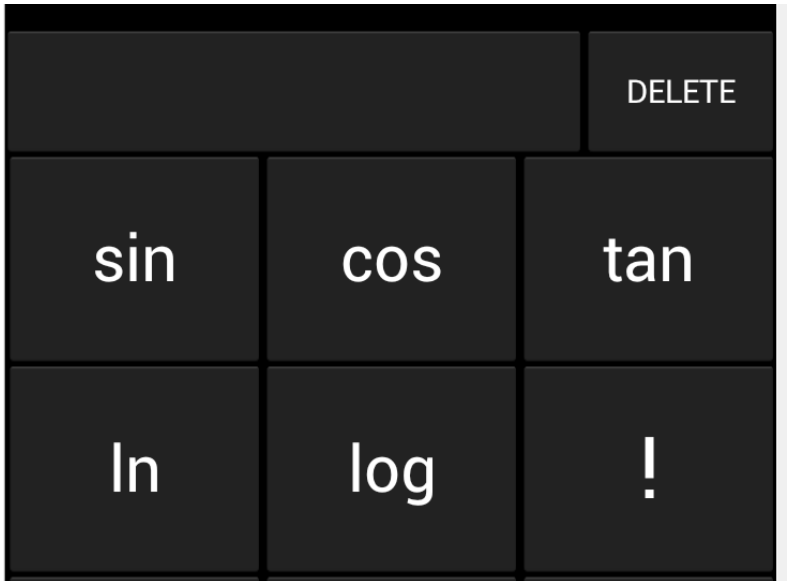
TextView

- Other possible attributes:
- set number of lines of text that are visible
  - android:lines="2"
- ellipssize attribute to add … instead of simply truncating text
- contextual links to email address, url, phone number,
  - autolink attribute set to none, web, email, phone, map, or all

Common Controls - Button

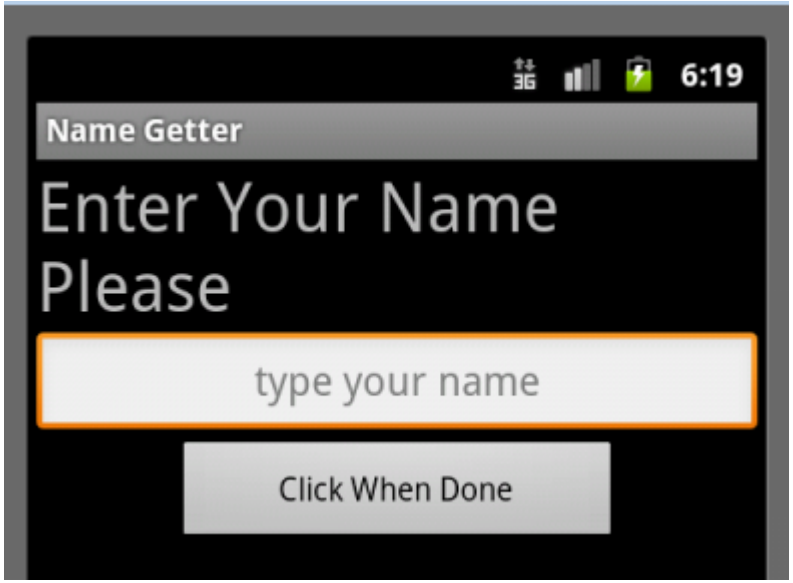
- Text or icon or both on View
- button press triggers some action
  - set android:onClick attribute in XML file
  - OR create a ClickListener object, override onClick method, and register it with the checkbox
    - typically done with anonymous inner class
  - possible to customize appearance of buttons

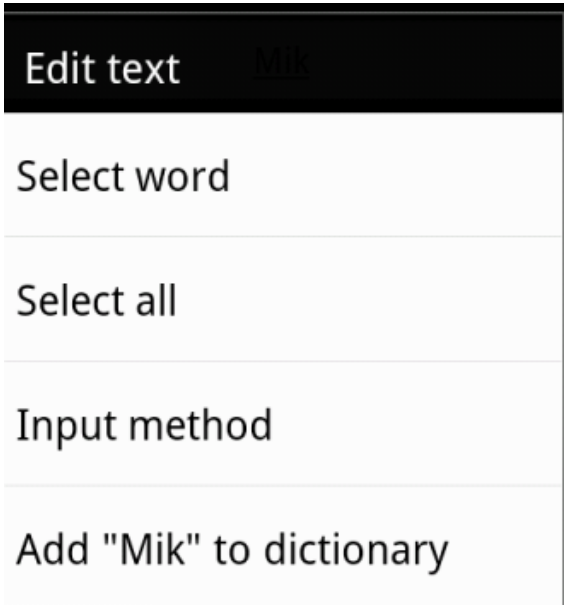
http://developer.android.com/guide/topics/ui/controls/button.html#CustomBackground



Common Controls - EditText

- Common component to get information from the user
- long press brings up context menu






```
<EditText
    android:id="@+id/edittext"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:gravity="center"
    android:inputType="textPersonName"
    android:hint="type your name" />
```

EditText

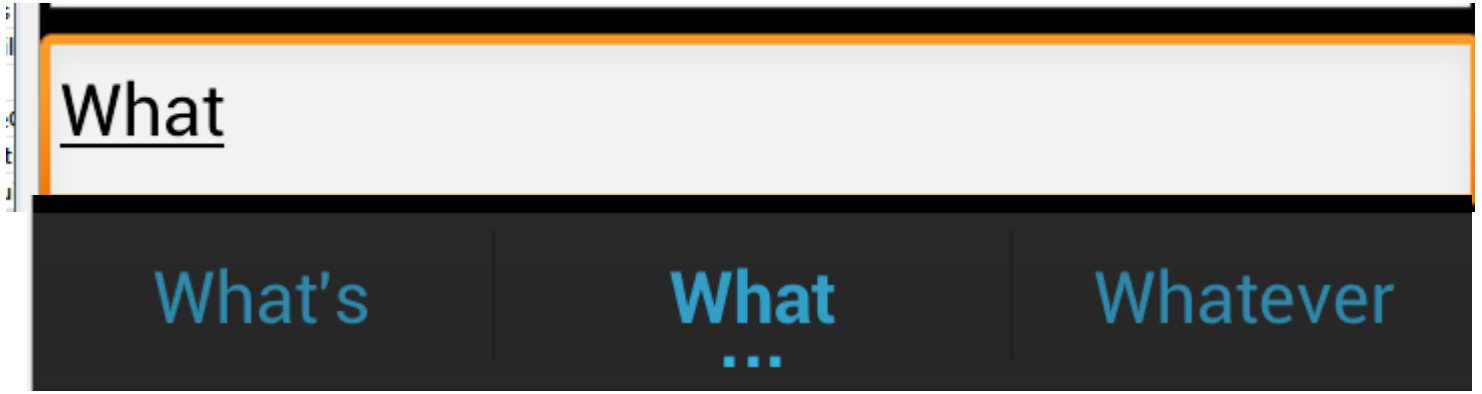
- can span multiple lines via android:lines attribute
- **Text fields can have different input types, such as number, date, password, or email address**
  - android:inputType attribute
  - affects what type of keyboard pops up for user and behaviors such as is every word capitalized

EditText

- Keyboard actions
  - specify action when input done
  - ime = input method editor
- android:imeOptions attribute
  - actionNone, actionSearch, actionSend, others
    - http://developer.android.com/reference/android/widget/TextView.html#attr_android:imeOptions

Auto Complete Options

- Depending on EditText inputType suggestions can be displayed
  - works on actual devices

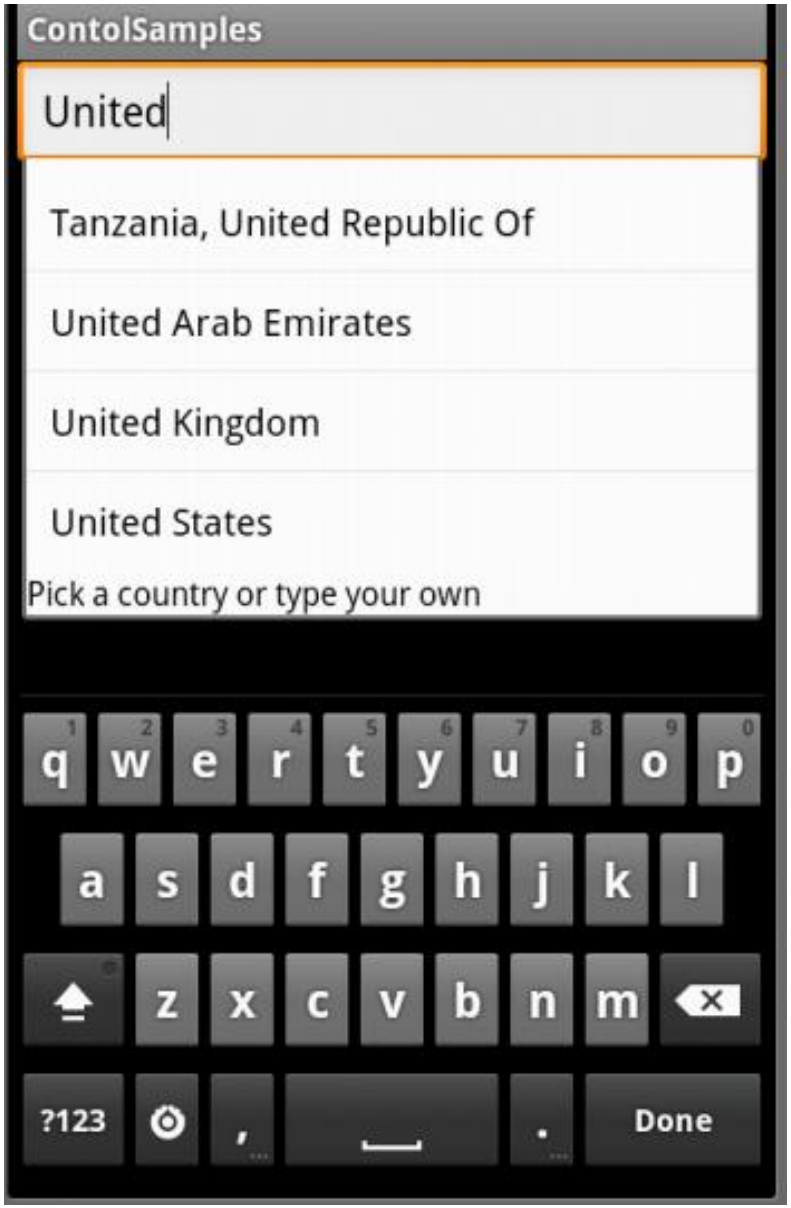


- Other classes exist for auto complete from list
  - AutoCompleteTextView
    - choose one option
  - MultiAutoCompleteTextView
    - choose multiple options (examples tags, colors)

AutoCompleteTextView

- Two types
  - we provide list of choices
  - user provides list
- Developer list
  - use ArrayAdapter connected to array
  - best practice: put array in array.xml file

AutoComplete Using Array



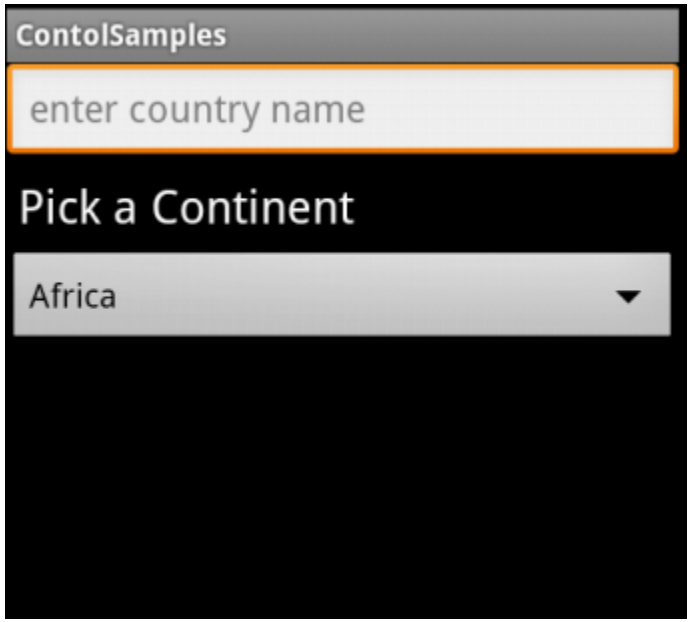
EditText

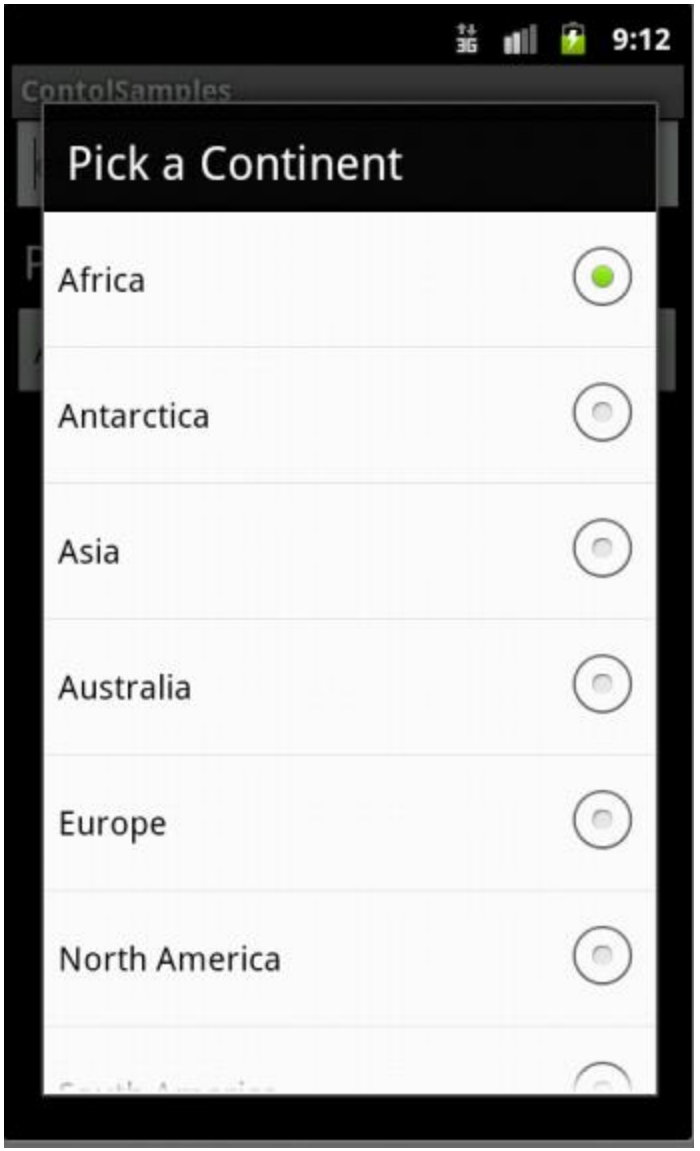
- Auto complete option using device dictionary:

```
<EditText
    android:id="@+id/msg_text_input"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:autoText="true"
    android:imeOptions="actionNone"
    android:text="" />
```

Spinner Controls

- Similar to auto complete, but user **must** select from a set of choices





Spinner Control

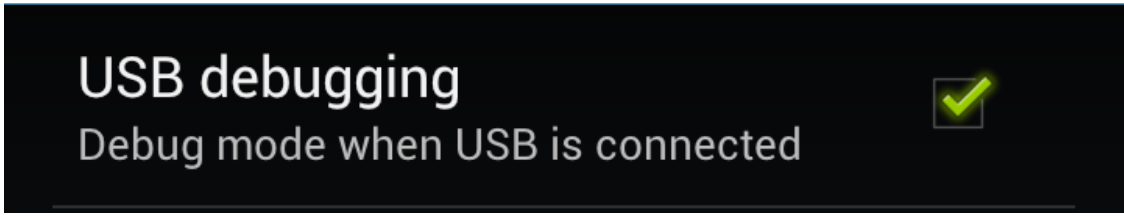
```
<Spinner
    android:id="@+id/spinner1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:entries="@array/continents"
    android:prompt="@string/pickCon"
    />
```

arrays.xml in res/values

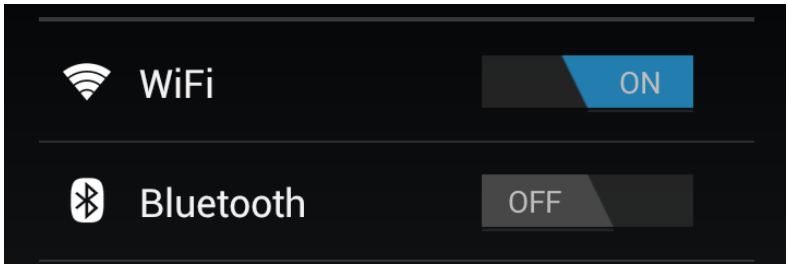
```
<string-array name="continents">
    <item>Africa</item>
    <item>Antarctica</item>
    <item>Asia</item>
    <item>Australia</item>
    <item>Europe</item>
    <item>North America</item>
    <item>South America</item>
</string-array>
```

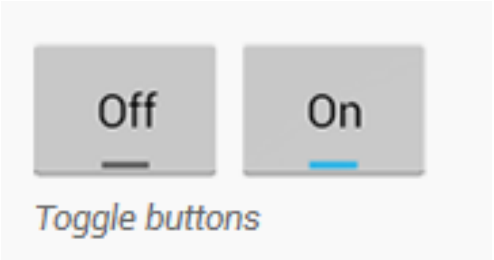
Simple User Selections

- CheckBox
  - set android:onClick attribute or create a ClickListener object, override onClick method, and register it with the checkbox



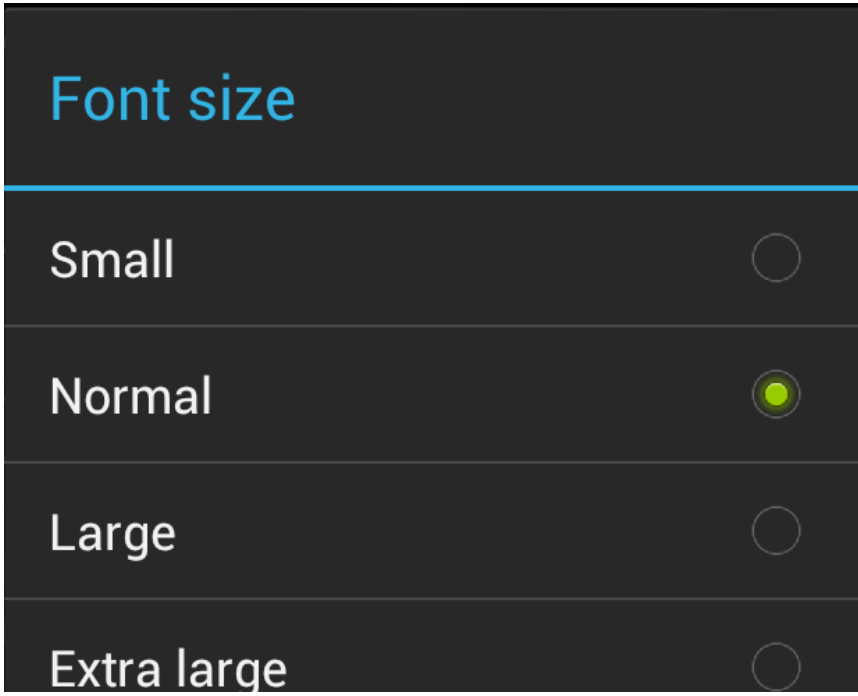
- Switches and ToggleButton
  - similar to CheckBox with two states, but visually shows states
  - on and off text





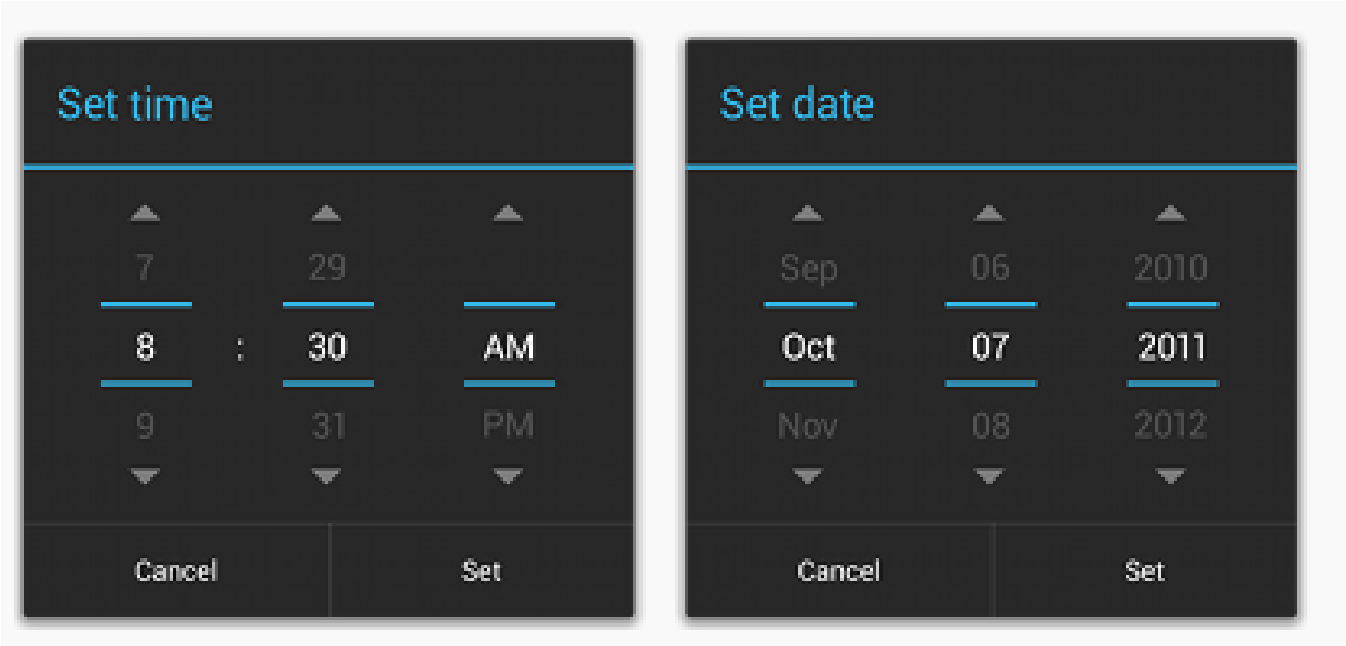
RadioButton and RadioGroup

- Select one option from a set
- set onClick method for each button
  - generally same method
- Collected in RadioGroup
  - sub class of LinearLayout
  - vertical or horizontal orientation



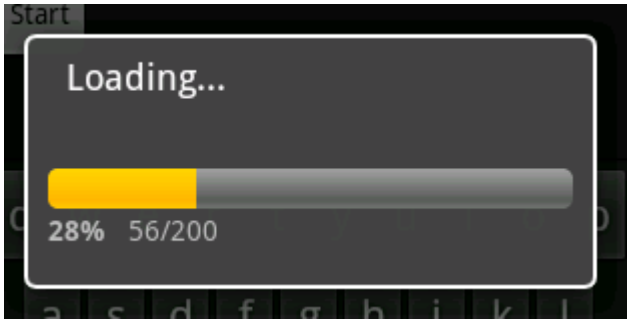
Pickers

- TimePicker and DatePicker
- Typically displayed in a TimePickerDialog or DatePickerDialog
  - dialogs are small windows that appear in front of the current activity

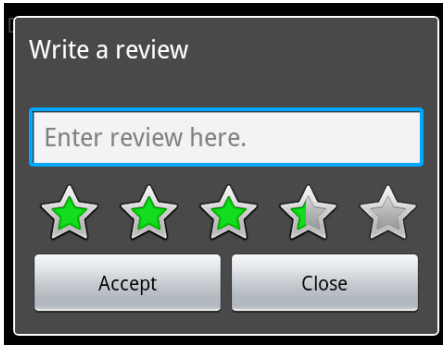


Indicators

- Variety of built in indicators in addition to TextView
- ProgressBar



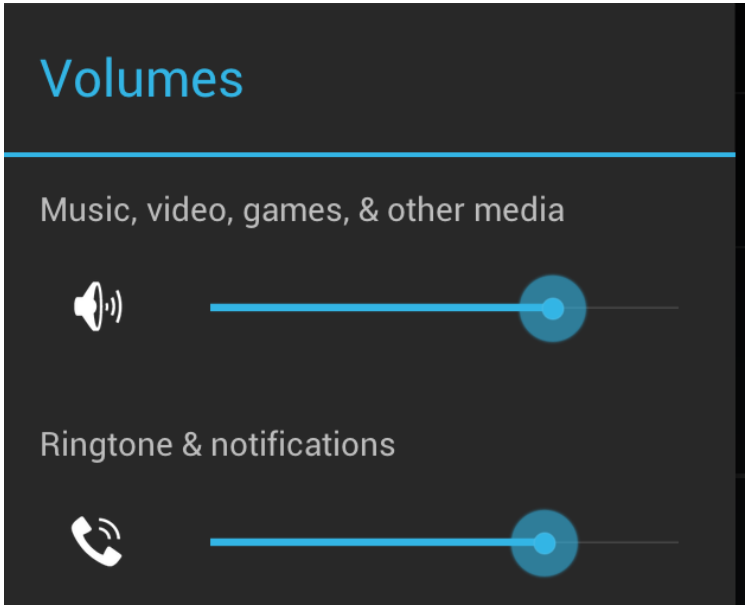
- RatingBar



- Chronometer
- DigitalClock
- AnalogClock

SeekBar

- a slider
- Subclass of progress bar
- implement a SeekBar.OnSeekBarChangeListener to respond to changes in setting



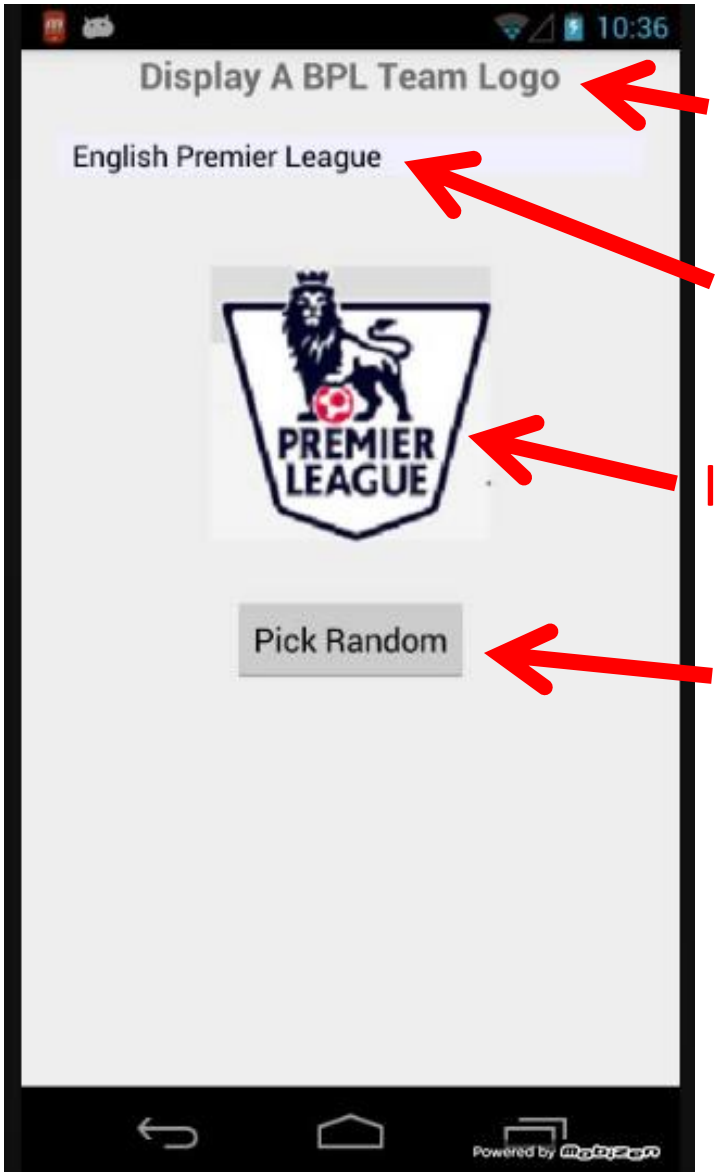
INTERACTING WITH WIDGETS

Interacting with Widgets

- Some widgets simply display information.
  - TextView, ImageView
- Many widgets respond to the user.
- We must implement code to respond to the user action.
- Typically we implement a listener and connect to the widget in code.
  - logic / response in the code

Example - Display Random Image

- App to display crests of British Premier League Football teams
- Allow user to select team from spinner control
- Or, press button to display a random crest



Button in XML layout file

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Pick Random"
    android:id="@+id/random_button"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="30dp" />
```

- Notice button reacts when pressed, but nothing happens
- Possible to disable button so it does not react

Responding to Button Press

- Two ways:
- Hard way, create a listener and attach to the button
  - shorter way exists for Views, but this approach is typical for many, many other widgets behaviors besides clicking
- Implement an onClickListener and attach to button

Accessing Button in Code

- R.java file automatically generated and creates ids for resources in project folder
  - if id attribute declared

```
/* 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 edu.utexas.scottm.bplteams;

public final class R {

    public static final class id {

        public static final int random_button=0x7f0c0042;
```

Setting Activity Layout / GUI

- Usually the GUI for an *Activity* is set in the onCreate method.
- Typically a layout file is used

```
public class BPL_Activity extends Activity {

 @Override
 protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_bpl_);
```

- set content view will *inflate* runtime objects for all the widgets in the layout file

Accessing Layout Widget

- To attach a listener we need a handle (reference) to the runtime object for the button (or desired widget)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_bpl_);
    getImageIDs();
    setSpinnerListener();
    setRandomButtonListener();
}
```

Accessing Layout Widget

```
private void setRandomButtonListener() {
    Button randomButton = (Button) findViewById(R.id.random_button);
```

- findViewById returns a View object
  - often necessary to cast to correct type
- A whole host of methods to access resources in your /res directory programmatically

Creating and attaching a Listener

```
randomButton.setOnClickListener(
        new View.OnClickListener() {
```

- setOnClickerListener is method that attaches the listener
- View.onClickListener is a Java interface with one method onClick
- We are implementing interface with an *anonymous inner class*

onClick Logic

```
@Override
public void onClick(View v) {
    // get the current selection
    Spinner spinner
            = (Spinner) findViewById(R.id.football_club_spinner);
    int oldIndex = spinner.getSelectedItemPosition();
    Log.d(TAG, "old index  = " + oldIndex);
    // don't want to pick the BPL symbol itself, so index 1 - 20
    int newIndex = randNumGen.nextInt(imageIDs.size() - 1) + 1;
    // don't let the new one be the old one
    // are we worried this will result in infinite loop with just
    while (oldIndex == newIndex) {
        newIndex = randNumGen.nextInt(imageIDs.size() - 1) + 1;
    }
    Log.d(TAG, "new index  = " + newIndex);
    ImageView iv = (ImageView) findViewById(R.id.imageView);
    iv.setImageResource(imageIDs.get(newIndex));
    spinner.setSelection(newIndex);
}
```

Shortcut for Clicks

- All View objects have an onClick attribute
- method to call when the View is clicked
- Can set onClick attribute to a method in Activity that is called when View is clicked

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Pick Random"
    android:id="@+id/random_button"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="30dp"
    android:onClick="pickRandom"/>
```

Shortcut for Clicks

- In Activity:

```
public void pickRandom(View v) {
    Spinner spinner
            = (Spinner) findViewById(R.id.football_club_spinner);
    int oldIndex = spinner.getSelectedItemPosition();
    Log.d(TAG, "old index  = " + oldIndex);
    // don't want to pick the BPL symbol itself, so index 1 - 20
    int newIndex = randNumGen.nextInt(imageIDs.size() - 1) + 1;
    // don't let the new one be the old one
    // are we worried this will result in infinite loop with just
    while (oldIndex == newIndex) {
        newIndex = randNumGen.nextInt(imageIDs.size() - 1) + 1;
    }
    Log.d(TAG, "new index  = " + newIndex);
    ImageView iv = (ImageView) findViewById(R.id.imageView);
    iv.setImageResource(imageIDs.get(newIndex));
    spinner.setSelection(newIndex);
}
```

- demo when method signature wrong

Clicker

- What method do we use to associate a variable with the runtime object of a UI component declared in a layout xml file?

A. setContentView()

B. startActivity()

C. onCreate()

D. a constructor

E. findViewById()

THEMES AND STYLES

Styles

- Attributes of a View can be set via to a Style
- A ***Style*** is a collection of attributes that specify the attributes and format of a View or window
- Styles defined in their own XML file and referenced by other views

Simplification via Styles

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="@string/hello" />
```

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textAppearance="@style/CodeFont"
    android:text="@string/hello" />
```

In separate XML file

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CodeFont" parent="@android:style/TextAppearance.Medium">
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

Themes

- Android defines themes which set default values for many, many attributes of widgets
- Themes have changed over time with different releases
  - theme
  - light
  - dark
  - material design
- Theme can be set in the Manifest file for the app

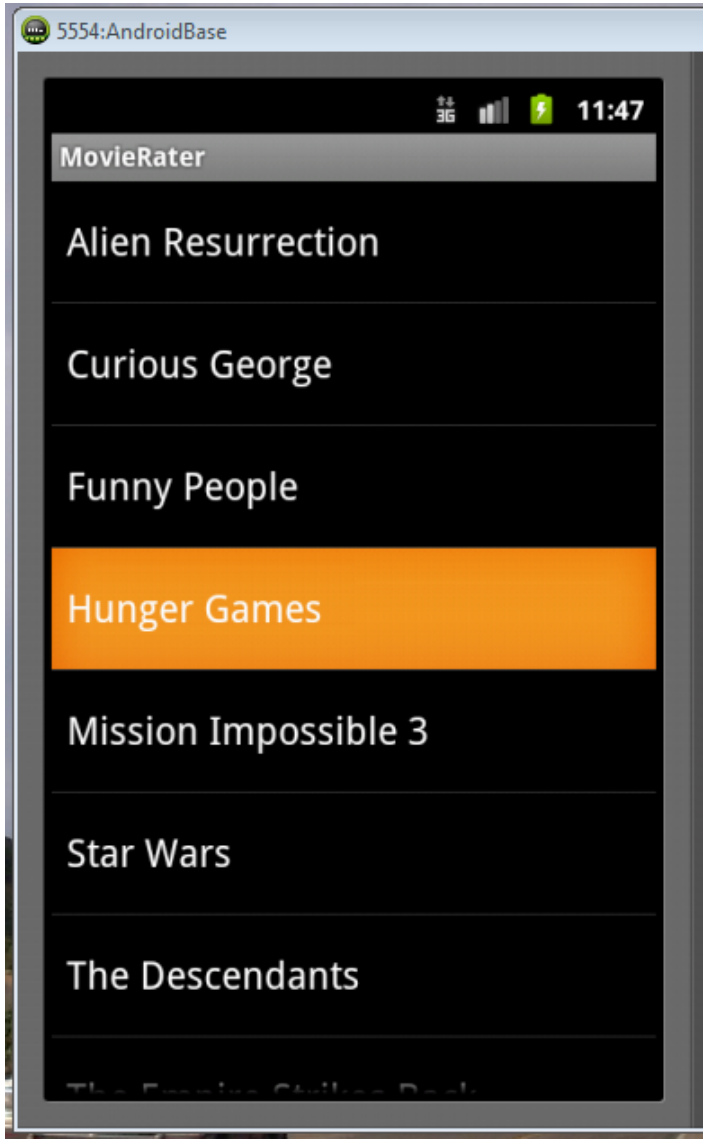
```
android:theme="@style/AppTheme">
```

DATA DRIVEN CONTAINERS

LISTVIEW AND GRIDVIEW

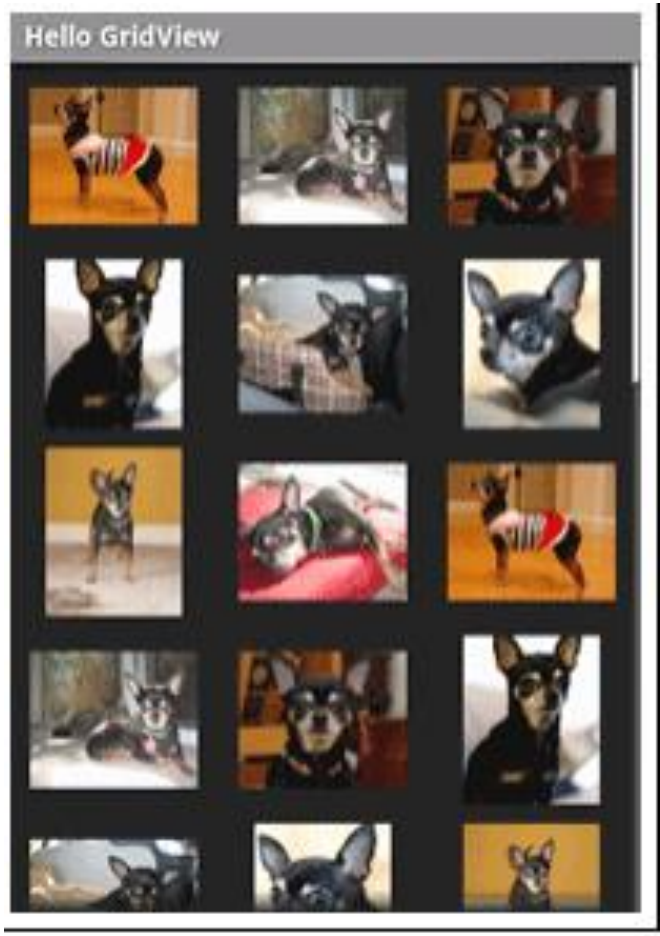
Data Driven Containers

- Containers that display repetitive child Views
- ListView
  - vertical scroll, horizontal row entries, pick item
  - consider using ListActivity
- GridView
  - specified number of rows and columns
- GalleryView
  - horizontal scrolling list, typically images



AdapterView

- ListView, GridView, and GalleryView are all sub classes of AdapterView
- Adapter generates child Views from some data source and populates the larger View
- Most common Adapters
  - CursorAdapter used when to read from database
  - ArrayAdapter to read from resource, typically an XML file



Adapters

- When using an Adapter a layout is defined for each child element (View)
- The adapter creates Views based on layout for each element in data source and fills the containing View (List, Grid, Gallery) with the created Views
  - binding
- child Views can be as simple as a TextView or more complex layouts / controls
  - simple ones provided in android.R.layout

Typical Adapter Example

```
public class CountryActivity extends ListActivity {

    private ListView view;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        ArrayAdapter<CharSequence> adapter
            = ArrayAdapter.createFromResource(this,
                    R.array.countries, R.layout.list_item);

        view = getListView();
        setListAdapter(adapter);
    }


}
```

Data Source - countries resource file

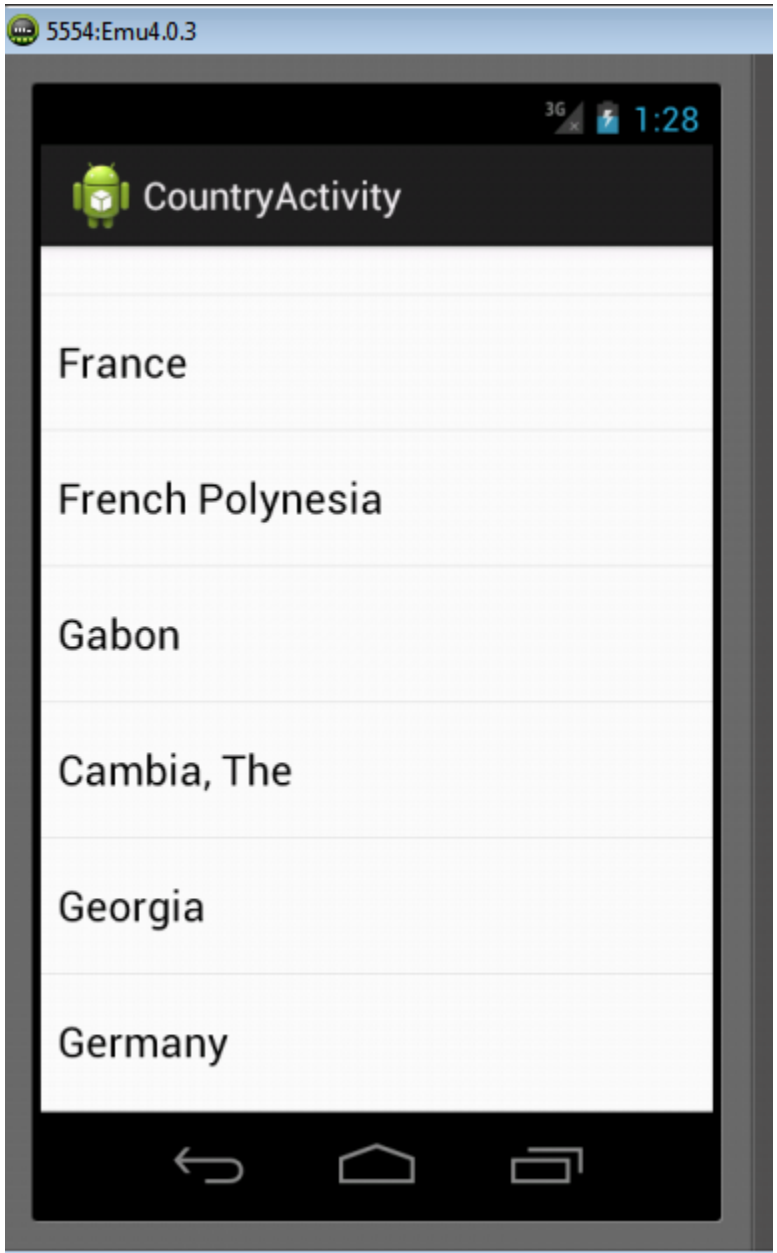
```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <array name="countries">
        <item>Abkhazia</item>
        <item>Afghanistan</item>
        <item>Akrotiri and Dhekelia</item>
        <item>Aland</item>
        <item>Albania</item>
        <item>Algeria</item>
        <item>American Samoa</item>
        <item>Andorra</item>
        <item>Angola</item>
        <item>Anguilla</item>
        <item>Antigua and Barbuda</item>
        <item>Argentina</item>
        <item>Armenia</item>
```

TextView for Data

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/countryTextView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_vertical"
    android:minHeight="?android:attr/listPreferredItemHeight"
    android:padding="8dp"
    android:textColor="@android:color/black"
    android:background="@android:color/white"
    android:textSize="20sp" >
</TextView>
```

- ListView filled with TextViews
- TextViews store data from ArrayAdapter

ListView and GridView Results



Selection Events

- ListView, GridView, GalleryView
- Typically user can select one item of data
- Implement the OnItemClickListener class and set it as the listener
  - we will do this a lot:
  - create a class that implements some kind of listener
  - register it with a control

Altering the Data and Display

- Previous example read data from resource file
- What if we want to update list view as data changes?
  - add and remove items
- Example: remove countries from list and view when selected

Altering Data

- ArrayAdapter serves as a bridge between a data source and a ListView
- Previous example, data was an array resource file
  - resource file won't change
- Dump data to List (ArrayList) and create ArrayAdapter from that source

Source Code

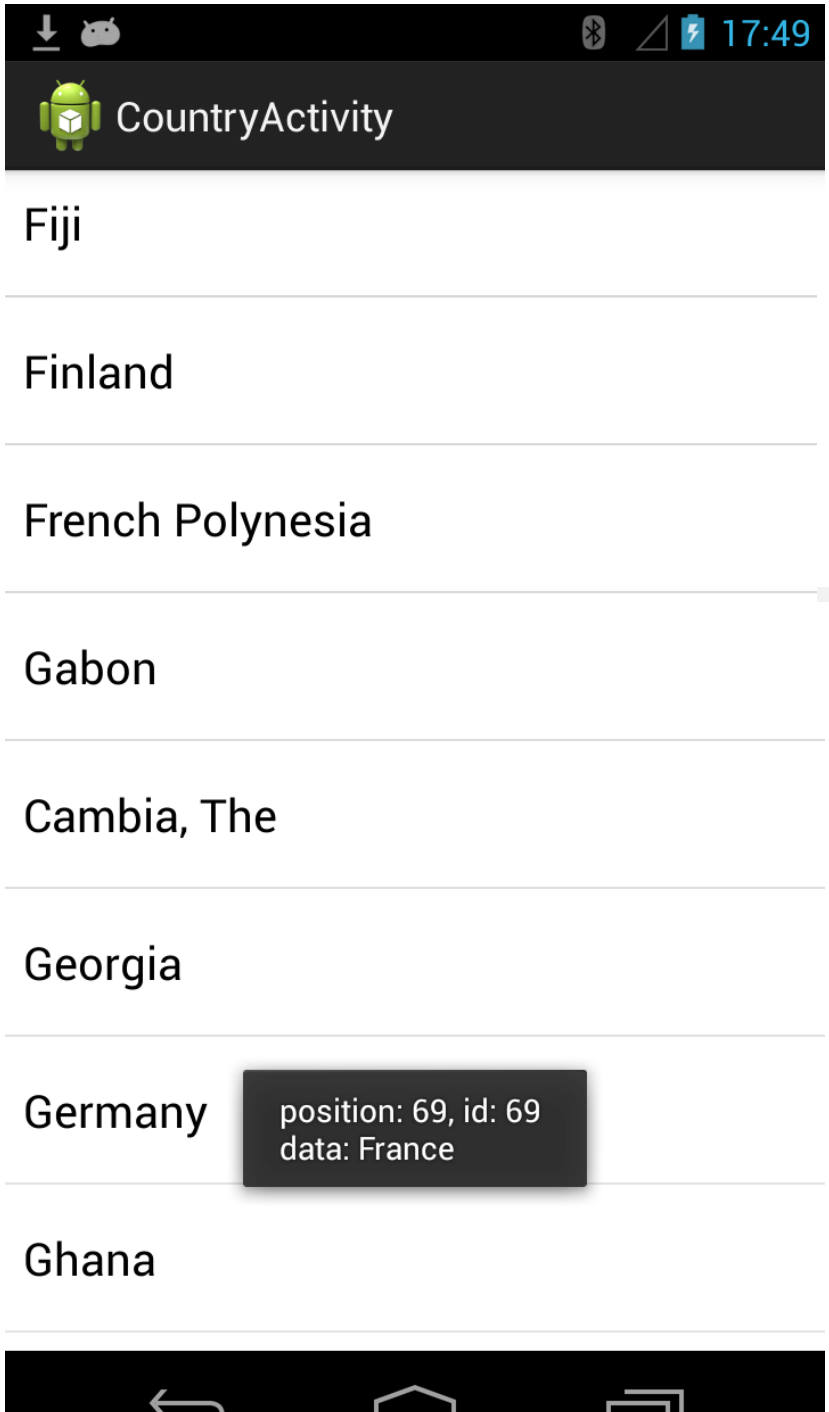
```
public class CountryActivity extends ListActivity {

    private ListView view;
    private ArrayList<String> countries;
    private ArrayAdapter<String> adapter;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        view = getListView();
        setAdapter();
```

Create ArrayList

```
private void setAdapter() {
    String[] rawData
        = getResources().getStringArray(R.array.countries);
    countries
        = new ArrayList<String>(Arrays.asList(rawData));
    adapter
        = new ArrayAdapter<String>(this, R.layout.list_item, countries)
    setListAdapter(adapter);
}
```

Alter Data on Select

```
view.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent,
                View v, int position, long id) {

        // remove item selected from arraylist
        countries.remove(position);

        adapter.notifyDataSetChanged();
        //view.invalidateViews();
```



A Toast

"A toast provides simple feedback about an operation in a small popup."

Creating a Toast

- Inside the OnItemClickListener anonymous inner class

```
Toast.makeText(CountryActivity.this,
        "position: " + position +
        ", id: " + id + "\ndata: "
        + countries.get(position),
        Toast.LENGTH_LONG).show();
```

More Complex List View Items

- What if we want each item in a list to have more than simple text?
- Let's add a switch to each ListView item to show if the Country listed is "safe" or not?
- Each View element in the list will be a horizontal linear layout with a TextView and a switch

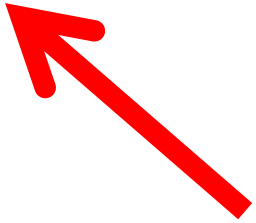
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/andro
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:descendantFocusability="blocksDescendants"
    android:orientation="horizontal" >

    <TextView
        android:id="@+id/countryTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:background="@android:color/white"
        android:gravity="center_vertical"
        android:minHeight="?android:attr/listPreferredItemHeight"
        android:padding="8dp"
        android:textColor="@android:color/black"
        android:textSize="20sp" >
    </TextView>

    <Switch
        android:id="@+id/countrySafeSwitch"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

Not all of layout file shown

Setting Adapter

- Change to use the complex layout for each ListView item

```
private void setAdapter() {
    // for layout with TextView in more complex layout
    adapter
            = new ArrayAdapter<String>(
            this, // context
            R.layout.complex_list_item, // layout of list items / ro
            R.id.countryTextView, // sub layout to place text
            countries); // model of text

    setListAdapter(adapter);
}
```

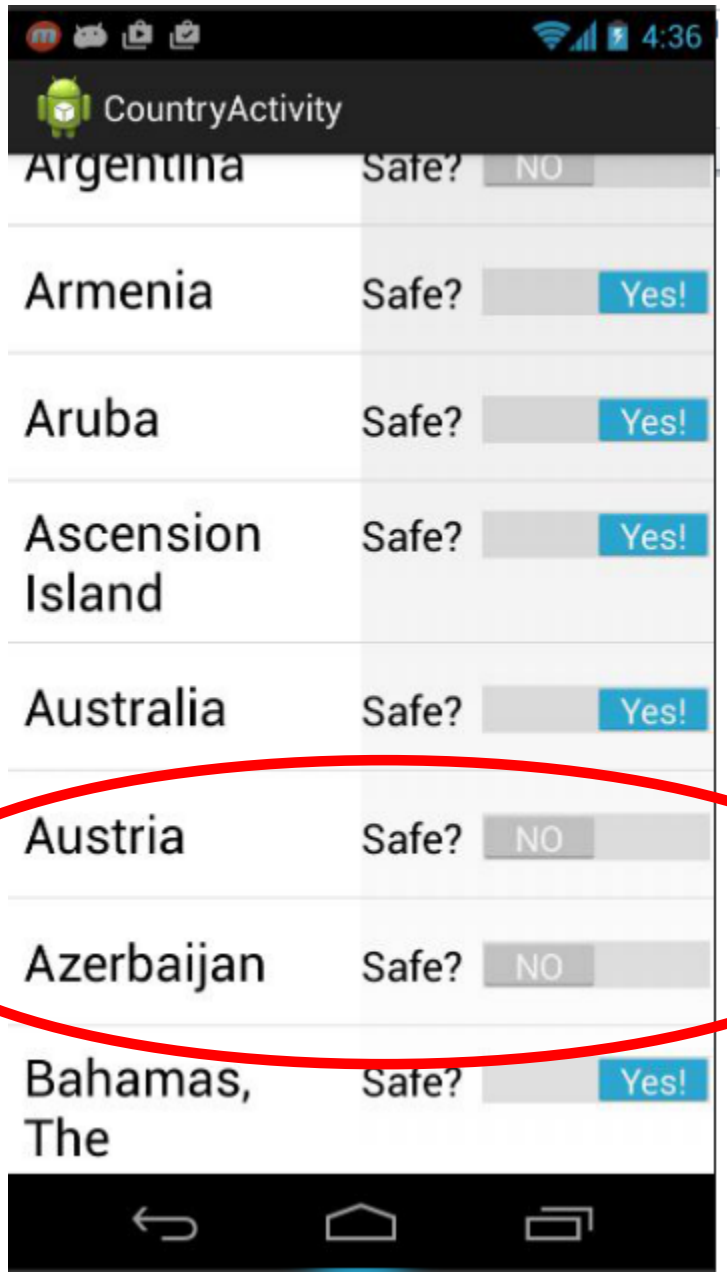


Result

- Looks okay.
- However...
- Scroll the list and notice all safe switches set to Yes!
- Flip a couple and scroll

View Recycling

Scroll



UH OH

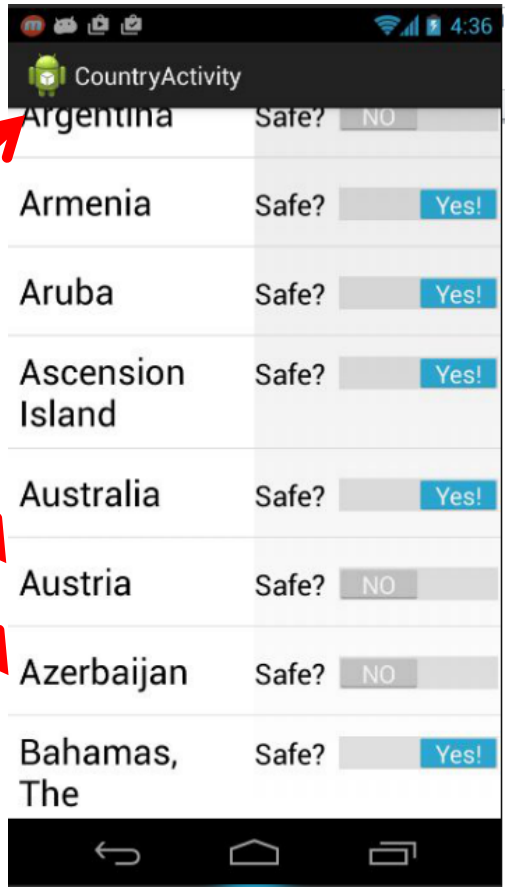
View Recycling

- Imagine a ListView tied to contacts on a phone or some other possibly large data set.
- Some people have 1000's of contacts.
- Creating a ListView with a distinct object for every list element (the Views) would require a LOT of memory.
- So, the rows in a list view get ***recycled***. Enough objects are created for the visible items, but as they scroll off the objects are reused and the data in the widgets is **reset** to what the user should see.

View Recycling

We set the switch on the row that contains Andorra to no. The we scrolled down the list. The List View item that contains Andorra is recycled.

The adapter we are using automatically alters the text, but the switch is still set to no!



Taking Control of Recycling

- We need to track the status of ***safe*** for each country and change the switch position as appropriate when a list view item gets recycled
- This requires creating two classes:
  - one to model the data for each row
  - our own Adapter that extends ArrayAdapter

CountryRowData

- Simple nested class to model and track the data in a row

```
private static class CountryRowData {
    private String name;
    private boolean safe;

    private CountryRowData(String n, boolean s) {
        name = n;
        safe = s;
    }

    public String toString() {
        return name;
    }
}
```

New onCreate Method

- Create list of CountryRowData objects and send to our new Adapter class

```
@Override
public void onCreate(Bundle bundle) {
    super.onCreate(bundle);
    ArrayList<CountryRowData> list
            = new ArrayList<CountryRowData>();
    String[] countries
            = getResources().getStringArray(R.array.countries);
    for (String s : countries) {
        list.add(new CountryRowData(s, true));
    }
    setListAdapter(new SafeAdapter(list));
}
```

Extending ArrayAdapter

```
private class SafeAdapter extends ArrayAdapter<CountryRowData> {

    SafeAdapter(ArrayList<CountryRowData> list) {
        super(CountryActivity.this,
                R.layout.complex_list_item,
                R.id.countryTextView,
                list);
    }

public View getView(int position, View convertView,
                    ViewGroup parent) {
    View row = super.getView(position, convertView, parent);
    Switch theSwitch =(Switch) row.getTag();
    if (theSwitch == null) {
        theSwitch = (Switch) row.findViewById(R.id.countrySafeSwitch
        row.setTag(theSwitch);
```

Listening for Changes to Switches

```
CompoundButton.OnCheckedChangeListener l =
        new CompoundButton.OnCheckedChangeListener() {
            public void onCheckedChanged(CompoundButton buttonView,
                                         boolean isChecked) {
                Integer myPosition = (Integer) buttonView.getTag();
                CountryRowData model = getModel(myPosition);
                model.safe = isChecked;
            }
        };
theSwitch.setOnCheckedChangeListener(l);
```

Set Switch to Correct Value

```
CountryRowData model = getModel(position);
theSwitch.setTag(position);
theSwitch.setChecked(model.safe);
return(row);
```

```
private CountryRowData getModel(int position) {
    return (((SafeAdapter) getListAdapter()).getItem(position));
}
```

Explanation of Adapter

- Our SafeAdapter class lets ArrayAdapter inflate and recycle the row
  - call to super.getView
  - this will set the country name
  - inflate = take an xml layout and create a runtime object to model it, measure and draw the object
- Then we check to see if we have a ***ViewHolder*** in the rows ***tag***.

Explanation of Adapter

- If we don't have a ViewHolder for the current row we create one and associate it with the row
- We add a switch listener for the switch in the row

ViewHolder and Tags

- All View objects (all GUI widgets are descendants of View) have a **setTag()** and **getTag()** method
- These methods allow us to associate an arbitrary object with the View (widget)
- The *holder pattern* uses the widget tag to hold an object which in turn holds each of child widgets of interest

ViewHolder and Tags

- The purpose of attaching a holder to the row Views is to avoid calling findViewById() again
  - can be slow

Recycling of ListView Elements

- LOOK HERE FOR intercepting the ListView items:
- http://stackoverflow.com/questions/6921462/listview-reusing-views-when-i-dont-want-it-to

Other Layouts - Tabbed Layouts

- Uses a TabHost and TabWidget
- TabHost consists of TabSpecs
- can use a TabActivity to simplify some operations
- Tabs can be
  - predefined View
  - Activity launched via Intent
  - generated View from TabContentFactory

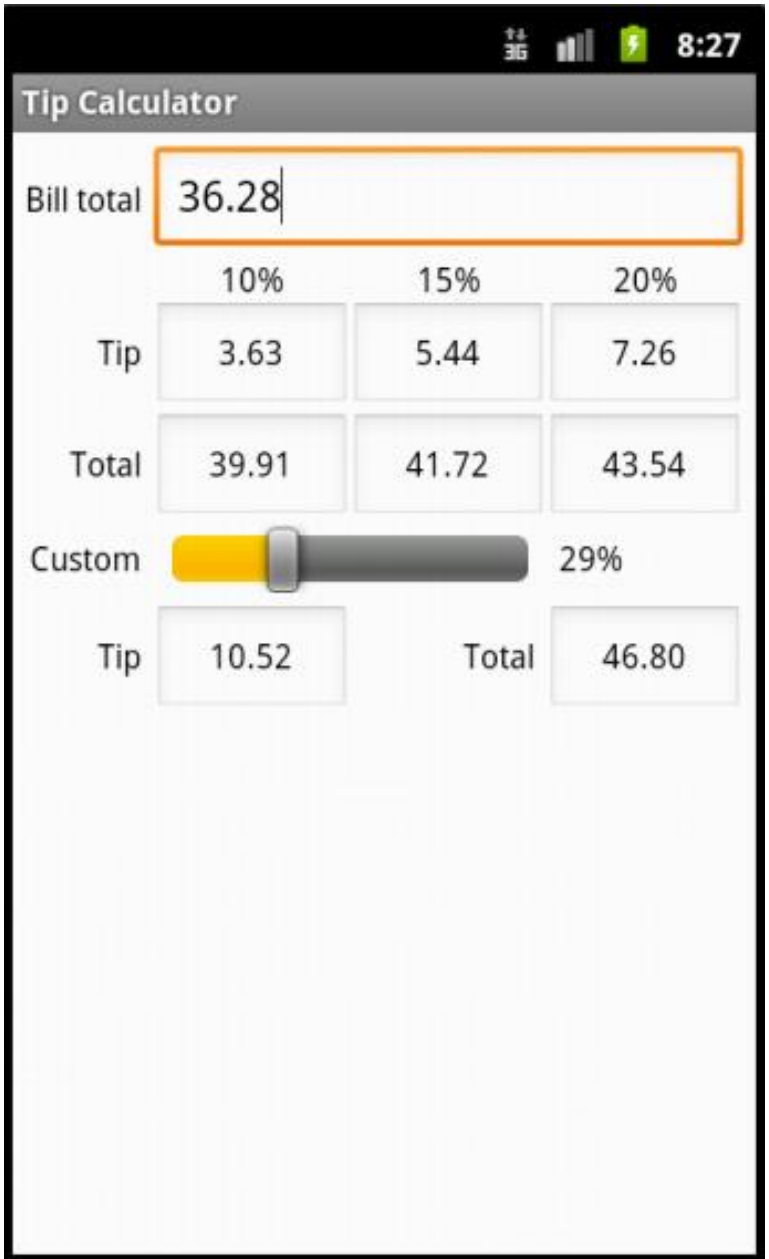
Scrolling

- ListView supports vertical scrolling
- Other views for Scrolling:
  - ScrollView for vertical scrolling
  - HorizontalScrollView
- Only one child View
  - but could have children of its own
- examples:
  - scroll through large image
  - Linear Layout with lots of elements

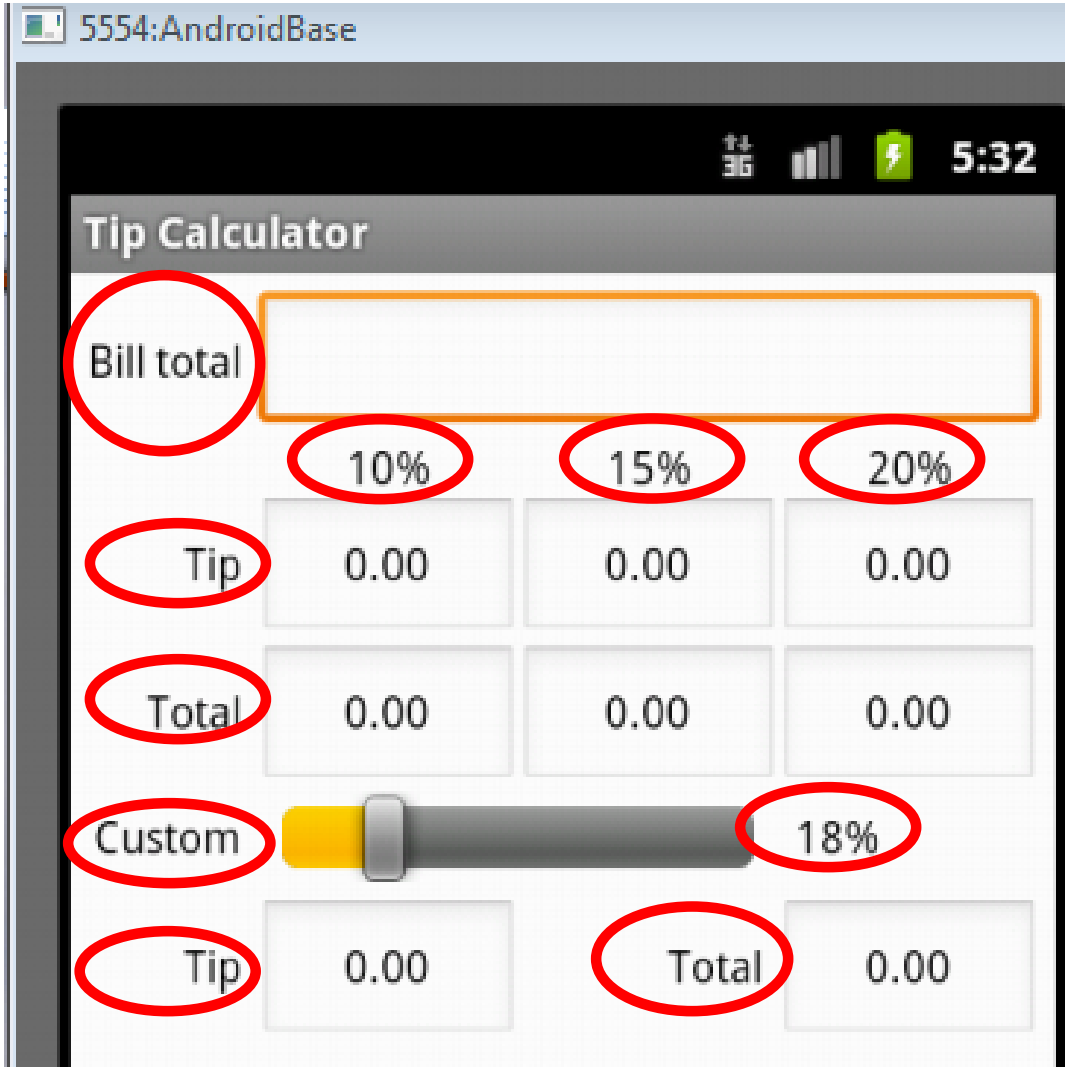
CONCRETE UI EXAMPLE - TIP CALCULATOR

Concrete Example

- Tip Calculator
- What kind of layout to use?
- Widgets:
  - TextView
  - EditText
  - SeekBar



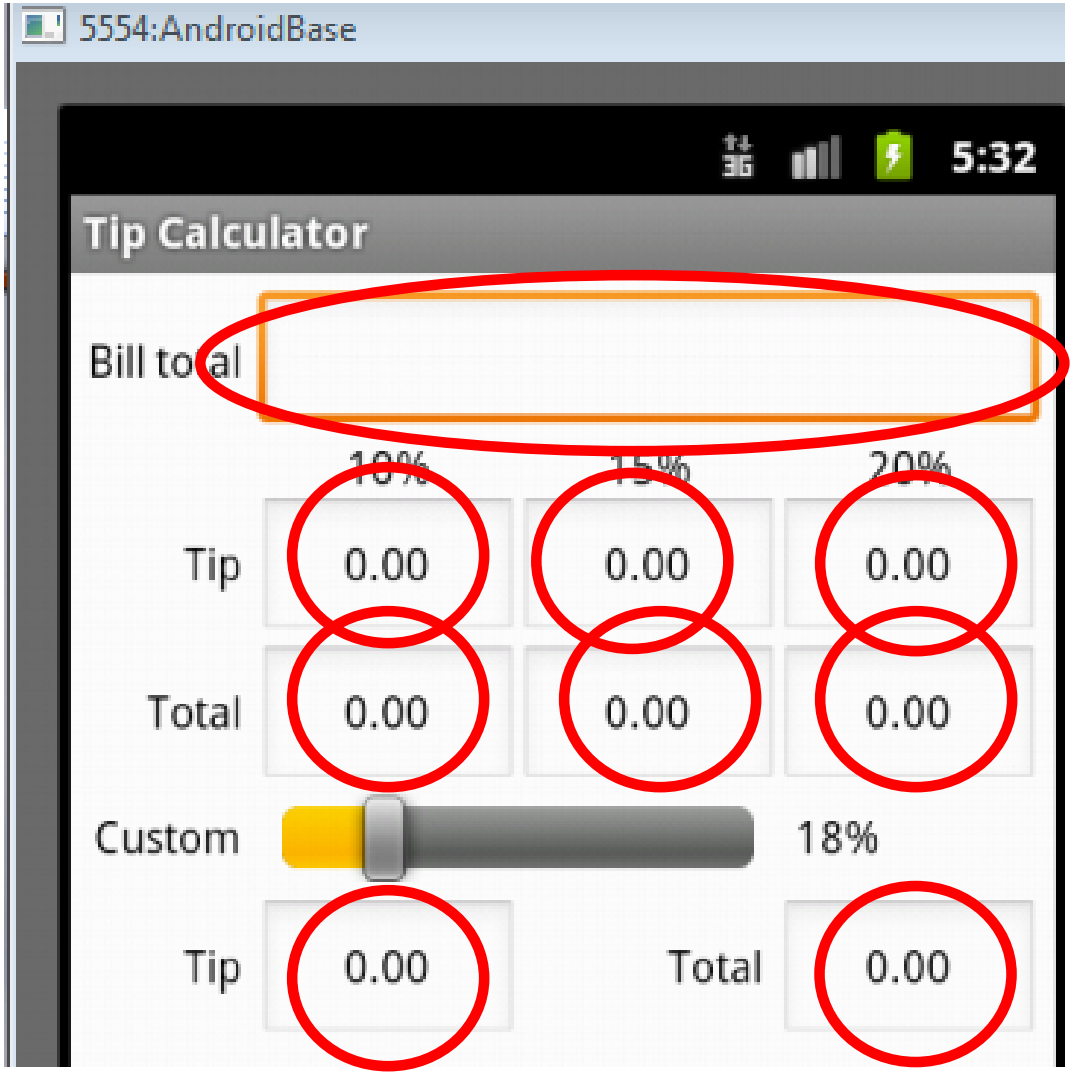
TextViews


5554:AndroidBase
5:32
Tip Calculator
Bill total
10%
15%
20%
Tip
0.00
0.00
0.00
Total
0.00
0.00
0.00
Custom
18%
Tip
0.00
Total
0.00

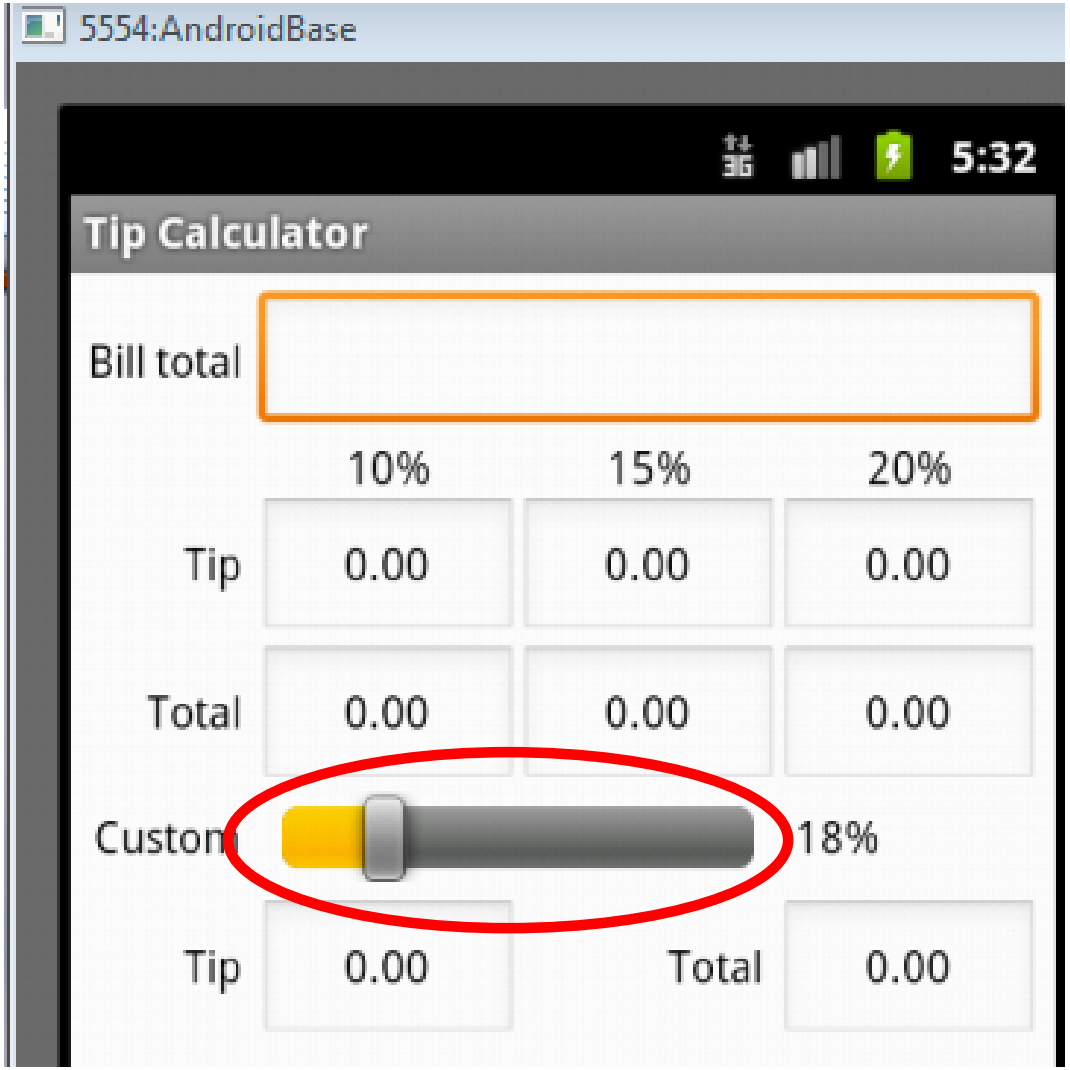
EditText

All but top EditText are uneditable

Alternative? TextViews?

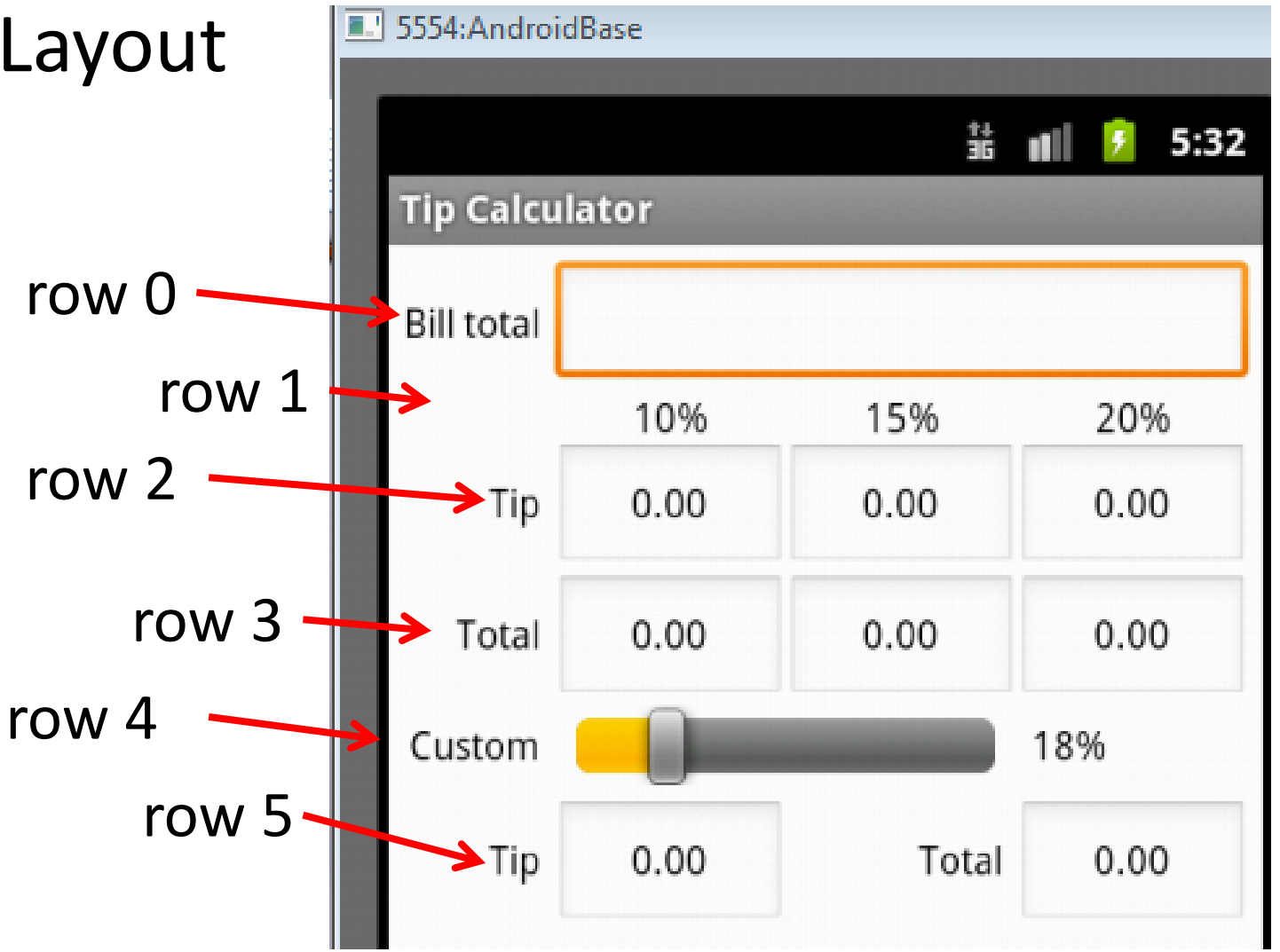


SeekBar


5554:AndroidBase
5:32
Tip Calculator
Bill total
10%
15%
20%
Tip
0.00
0.00
0.00
Total
0.00
0.00
0.00
Custom
18%
Tip
0.00
Total
0.00

Layout

- TableLayout



Layout Attributes

```
<TableLayout xmlns:android="http://sch
    android:id="@+id/tableLayout"
    android:layout_width="match_parent
    android:layout_height="match_paren
    android:background="#FFF"
    android:padding="5dp"
    android:stretchColumns="1,2,3" >
```

- android:background
  - #RGB, #ARGB, #RRGGBB, #AARRGGBB
  - can place colors in res/values/colors.xml

Color Resources

main.xml | colors.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3     <color name="Cardinal">#C41E3A</color>
4     <color name="White">#FFFFFF</color>
5 </resources>
```

```
android:layout_width="match_parent"
android:layout_height="match_parent"
android:background="@color/White"
android:padding="5dp"
```

- Good Resource / W3C colors
  - http://tinyurl.com/6py9huk

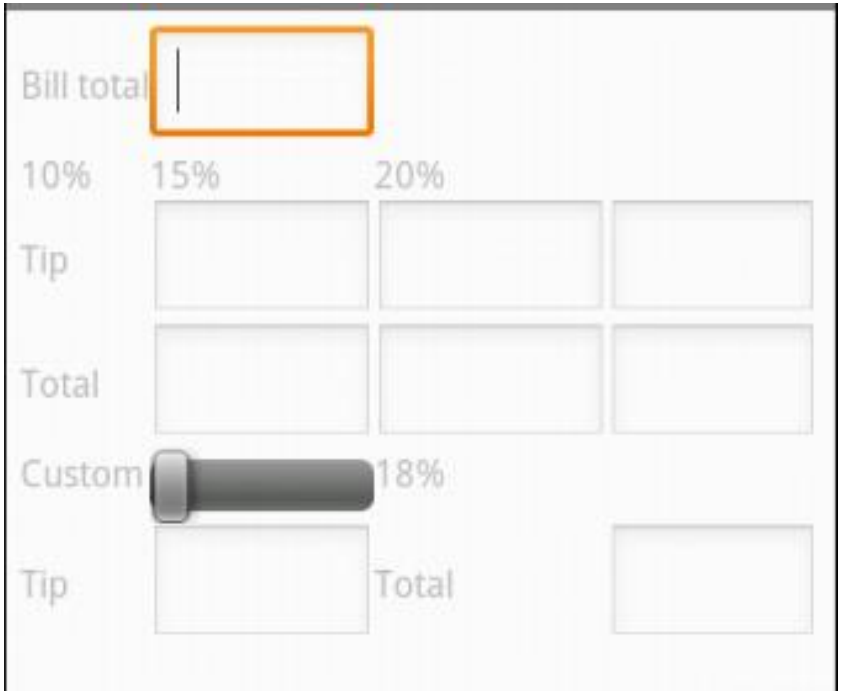
StretchColumns

```
<TableLayout xmlns:android="http://sch
    android:id="@+id/tableLayout"
    android:layout_width="match_parent
    android:layout_height="match_paren
    android:background="#FFF"
    android:padding="5dp"
    android:stretchColumns="1,2,3" >
```

- columns 0 indexed
- columns 1, 2, 3 stretch to fill layout width
- column 0 wide as widest element, plus any padding for that element

Initial UI

- Done via some Drag and Drop, Outline view, and editing XML
- Demo outline view
  - properties

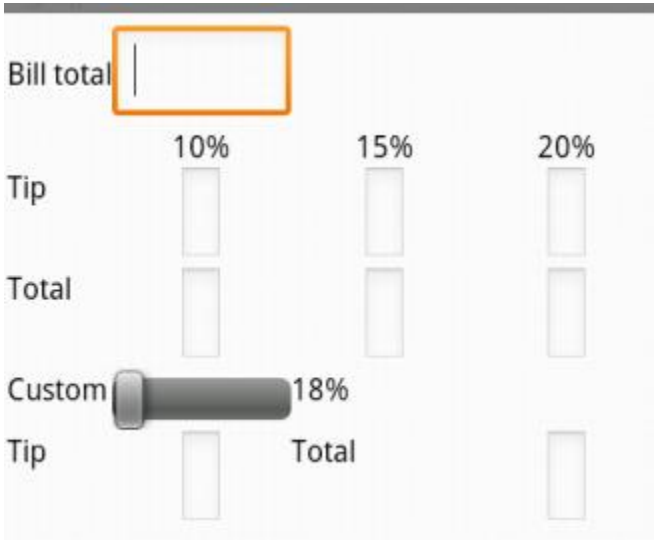


Changes to UI

- Outline multiple select properties
  - all TextViews' textColor set to black #000000
- change column for %DD labels

```
android:text="10%"
android:layout_column="1"
android:textColor="#000000" />
```

- use center gravity for components



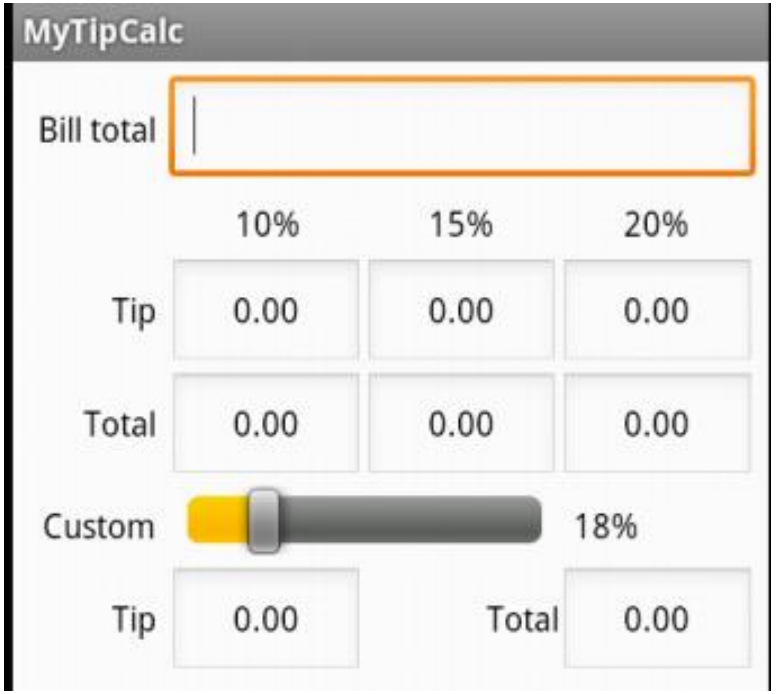
Changes to UI

- change bill total and seekbar to span more columns
- gravity and padding for text in column 0
- align text with seekBar
- set seekBar progress to 18
- set seekBar focusable to false - keep keyboard on screen

```
<EditText
    android:id="@+id/billEditText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_span="3"
    android:inputType="numberDecimal" >
```

Changes to UI

- Prevent Editing in EditText
  - focusable, long clickable, and cursor visible properties to false
- Set text in EditText to 0.00
- Change weights to 1 to spread out



Functionality

- onCreate instance variables assigned to components found via ids
- update standard percents:

```
private void updateStandard()
{

    for(int i = 0; i < NUM_PERCENTS - 1; i++) {
        double tip = currentBillTotal * tipPercents[i];
        double total = currentBillTotal + tip;
        tipEditTexts[i].setText(String.format("%.02f", tip));
        totalEditTexts[i].setText(String.format("%.02f", total));
    }

} // end method updateStandard
```

Functionality - Saving State

- onSaveInstance
  - save BillTotal and CustomPercent to the Bundle
  - check for these in onCreate

```
// save values of billEditText and customSeekBar
@Override
protected void onSaveInstanceState(Bundle outState)
{
   super.onSaveInstanceState(outState);

   outState.putDouble(BILL_TOTAL, currentBillTotal);
   outState.putInt(CUSTOM_PERCENT, (int) (tipPercents[CUSTOM_INDEX] * 100));
} // end method onSaveInstanceState
```

Functionality Responding to SeekBar

- customSeekBarListener instance variable
- Of type OnSeekBarChangeListener

public static interface

SeekBar.OnSeekBarChangeListener

Public Methods	
abstract void	onProgressChanged (SeekBar seekBar, int progress, boolean fromUser) Notification that the progress level has changed.
abstract void	onStartTrackingTouch (SeekBar seekBar) Notification that the user has started a touch gesture.
abstract void	onStopTrackingTouch (SeekBar seekBar) Notification that the user has finished a touch gesture.

Create an Anonymous Inner Class

- Class notified when seek bar changed and program updates custom tip and total amount
- must register with the seekBar instance variable in onCreate!

```
// called when the user changes the position of SeekBar
private OnSeekBarChangeListener customSeekBarListener =
   new OnSeekBarChangeListener()
{
   // update tipPercents[CUSTOM_INDEX], then call updateCustom
   @Override
   public void onProgressChanged(SeekBar seekBar, int progress,
      boolean fromUser)
   {
      // sets tipPercents[CUSTOM_INDEX] to position of the Seek
      tipPercents[CUSTOM_INDEX] = seekBar.getProgress();
      updateCustom(); // update EditTexts for custom tip and to
   }
```

Functionality - Total EditText

public interface
TextWatcher

Public Methods	
abstract void	afterTextChanged (Editable s) This method is called to notify you that, somewhere within `s`, the text has been changed.
abstract void	beforeTextChanged (CharSequence s, int start, int count, int after) This method is called to notify you that, within `s`, the `count` characters beginning at `start` are about to be replaced by new text with length `after`.
abstract void	onTextChanged (CharSequence s, int start, int before, int count) This method is called to notify you that, within `s`, the `count` characters beginning at `start` have just replaced old text that had length `before`.

- Another anonymous inner class
- implement onTextChanged to converts to double and call update methods
- register with EditText for total in onCreate()!