CS371m - Mobile Computing

User Interface Basics
UI Programming with Widgets

• Widget is 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
Widgets

• Including:
• Text Views
• 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
Containers

• Containers are used to organize multiple widgets into a structure
• Similar to layout managers in Java
• aka ViewGroups
• Containers have children
• Children can be UI widgets or other containers
• Containers have a set of rules governing how it lays out its children in the screen space the container occupies
Containers (ViewGroups) and Widgets (Views)

- TextViews (labels)
- ImageViews
- Controls such as buttons, etc.

A layout, for example a linear layout

A layout, for example a table layout
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.
Size - Wrap Content

```xml
<Button
    android:id="@+id/clickForActivityButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
>
```
<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
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](http://tinyurl.com/y8jj5eo)

• attributes can be set in the xml and most can changed programmatically
## Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Related Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>android:baselineAligned</td>
<td>setBaselineAligned(boolean)</td>
<td>When set to false, prevents the layout from aligning its children's baselines.</td>
</tr>
<tr>
<td>android:baselineAlignedChildIndex</td>
<td>setBaselineAlignedChildIndex(int)</td>
<td>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).</td>
</tr>
<tr>
<td>android:divider</td>
<td>setDividerDrawable(Drawable)</td>
<td>Drawable to use as a vertical divider between buttons.</td>
</tr>
<tr>
<td>android:gravity</td>
<td>setGravity(int)</td>
<td>Specifies how to place the content of an object, both on the x- and y-axis, within the object itself.</td>
</tr>
<tr>
<td>android:measureWithLargestChild</td>
<td>setMeasureWithLargestChildEnabled(boolean)</td>
<td>When set to true, all children with a weight will be considered having the minimum size of the largest child.</td>
</tr>
<tr>
<td>android:orientation</td>
<td>setOrientation(int)</td>
<td>Should the layout be a column or a row? Use &quot;horizontal&quot; for a row, &quot;vertical&quot; for a column.</td>
</tr>
<tr>
<td>android:weightSum</td>
<td></td>
<td>Defines the maximum weight sum.</td>
</tr>
</tbody>
</table>

### Inherited XML Attributes

From class `android.view.ViewGroup`

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

```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/clickForActivityButtonTitle"
    android:textColor="#FFF"
    android:textSize="30sp" />
```

Programmatically in Activity (Java code)

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

```xml
<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](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"
  - ellipsize 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

```xml
<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
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:

```xml
<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=""
/>```

• No dictionary on emulator!
Spinner Controls

• Similar to auto complete, but user **must** select from a set of choices
Spinner Control

```xml
<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

```xml
<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

![Image of time and date pickers](image-url)
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
CONTAINERS FOR WIDGETS
VIEW GROUPS
ViewGroups - Layouts

• Layouts are subclasses of ViewGroup
• Still a view but doesn't actually draw anything
• serves as a container for other views
  – similar to Java layout managers
• options on how sub views (and view groups) are arranged
• Useful Layouts: FrameLayout, LinearLayout, TableLayout, GridLayout, RelativeLayout, ListView, GridView, ScrollView, DrawerLayout, ViewPager
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
Activity and Layout

• Code on previous slide demonstrates inflating a layout for an Activity
• ... and programmatically changing an attribute of the layout (or view) during runtime
Modifying Attributes

- in xml, programmatically, and visual editor
Gravity Attribute

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

center
right
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 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>
<RelativeLayout>

<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
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
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);
    }
}
<?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>
  </array>
</resources>
TextView for Data

```xml
<?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

- France
- French Polynesia
- Gabon
- Cambia, The
- Georgia
- Germany

GridViewTest

<table>
<thead>
<tr>
<th>Abkhazia</th>
<th>Afghanistan</th>
<th>Akrotiri and Dhekelia</th>
<th>Aland</th>
<th>Albania</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algeria</td>
<td>American Samoa</td>
<td>Andorra</td>
<td>Angola</td>
<td>Anguilla</td>
</tr>
<tr>
<td>Antigua and Barbuda</td>
<td>Argentina</td>
<td>Armenia</td>
<td>Aruba</td>
<td>Ascension Island</td>
</tr>
<tr>
<td>Australia</td>
<td>Austria</td>
<td>Azerbaijan</td>
<td>Bahamas, The</td>
<td>Bahrain</td>
</tr>
<tr>
<td>Bangladesh</td>
<td>Barbados</td>
<td>Belarus</td>
<td>Belgium</td>
<td>Belize</td>
</tr>
<tr>
<td>Benin</td>
<td>Bermuda</td>
<td>Bhutan</td>
<td>Bolivia</td>
<td>Bosnia and Herzegovina</td>
</tr>
<tr>
<td>Botswana</td>
<td>Brazil</td>
<td>Brunei</td>
<td>Bulgaria</td>
<td>Burkina</td>
</tr>
</tbody>
</table>
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
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

```java
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);
}
```
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
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        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"
Setting Adapter

- Change to use the complex layout for each ListView item

```java
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 / root
        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
<table>
<thead>
<tr>
<th>Country</th>
<th>Safe?</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>American Samoa</td>
<td>Yes!</td>
<td></td>
</tr>
<tr>
<td>Andorra</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Angola</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Anguilla</td>
<td>Yes!</td>
<td></td>
</tr>
<tr>
<td>Antigua and Barbuda</td>
<td>Yes!</td>
<td></td>
</tr>
<tr>
<td>Argentina</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Armenia</td>
<td>Yes!</td>
<td></td>
</tr>
<tr>
<td>Aruba</td>
<td>Yes!</td>
<td></td>
</tr>
<tr>
<td>Ascension Island</td>
<td>Yes!</td>
<td></td>
</tr>
<tr>
<td>Australia</td>
<td>Yes!</td>
<td></td>
</tr>
<tr>
<td>Austria</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Azerbaijan</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Bahamas, The</td>
<td>Yes!</td>
<td></td>
</tr>
</tbody>
</table>
View Recycling

- Imagine a ListView tied to contacts on a phone or some other possible 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 is 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 simply reset to what the user should see.
We set the switch on the row that contains Andorra to no. Then 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

```java
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

```java
@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

```java
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

```java
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;
                LinearLayout parent = (LinearLayout) buttonView.getParent();
                TextView label =
                    (TextView) parent.findViewById(R.id.countryTextView);
                label.setText(model.toString());
            }
        };
theSwitch.setOnCheckedChangeListener(l);
```
Set Switch to Correct Value

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

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
All but top EditText are uneditable

Alternative? TextViews?
SeekBar
• **TableLayout**

  row 0
  row 1
  row 2
  row 3
  row 4
  row 5
Layout Attributes

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

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

```xml
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/tableLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    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

```xml
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
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:

```java
private void updateStandard() {

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

} // end method updateStandard
```
Functionality - Saving State

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

```java
// 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) (tipPercent[CUSTOM_INDEX] * 100));
} // end method onSaveInstanceState
```
Functionality Responding to SeekBar

• customSeekBarChangeListener instance variable
• Of type OnSeekBarChangeListener

**public static interface SeekBar.OnSeekBarChangeListener**

<table>
<thead>
<tr>
<th>Public Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract void onProgressChanged</td>
<td>Notification that the progress level has changed.</td>
</tr>
<tr>
<td>onStartTrackingTouch</td>
<td>Notification that the user has started a touch gesture.</td>
</tr>
<tr>
<td>onStopTrackingTouch</td>
<td>Notification that the user has finished a touch gesture.</td>
</tr>
</tbody>
</table>
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!

```java
// 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

<table>
<thead>
<tr>
<th>Public Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract void afterTextChanged (Editable s)</td>
</tr>
<tr>
<td>This method is called to notify you that, somewhere within s, the text has been changed.</td>
</tr>
<tr>
<td>abstract void beforeTextChanged (CharSequence s, int start, int count, int after)</td>
</tr>
<tr>
<td>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.</td>
</tr>
<tr>
<td>abstract void onTextChanged (CharSequence s, int start, int before, int count)</td>
</tr>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

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