Google Android Development

Lesson #5

Preparation

You should never jump into coding when gearing up to work on an Android app (any app, for that matter). The steps you should take are outlined in the next several slides, but generally are:

- Determine what your app should do (Requirements)
- Build a design
- Write code
- Test
- Release

Requirements

Figure out the exact functionality that your app must perform. This sounds trivial because you know what you want your app to do, except that when you start working out the details, you'll find often enough, that you have overlooked something critical.

By getting the requirements down on paper first, you can be certain that you didn't miss anything.

Design

I like using various mock-up tools to design the GUI for my apps. I use Balsamiq Mock-up for PC / OS X, and MockUps on the iPad.

Both applications let me play around with various elements and help me position things the way I want.

The designs you build, and the requirements you have, become the blueprints for your applications.

Code, Test, and Release

With the design done, you're ready to code your application.

Thanks to the design work you did up front, it is now easier to build the XML GUI layouts, since you already have a pretty firm idea of what you want, and how to achieve it.

The requirements is helping you get the logic of your application correct as well.

Once the app is built, test it on various AVDs for different resolutions, and densities. Once done, get it onto a real device and test it there as well.

Finally, launch the app into the Android Marketplace (covered in Day 7).

Android GUI Design

Probably one of the more difficult chores when building an Android app, is the ability to create a compelling graphical user interface (GUI) for your users to work with. Your GUI needs to be simple, finger-clickable, easily understood, and functional. It should also be appealing to the eye.

As you have seen, Google does provide a basic tool for GUI design within Eclipse. As an alternative, there is a tool called DroidDraw, which is available for Windows, OS X, and Linux (DroidDraw.org).

Unfortunately, neither the Eclipse implementation nor DroidDraw hit the mark for ease of use, and functionality. It will probably be several years before we see a drastic improvement in this area, but eventually, I would like to see a design tool for Android similar to how Microsoft lets us create Silverlight applications for Windows Phone 7.

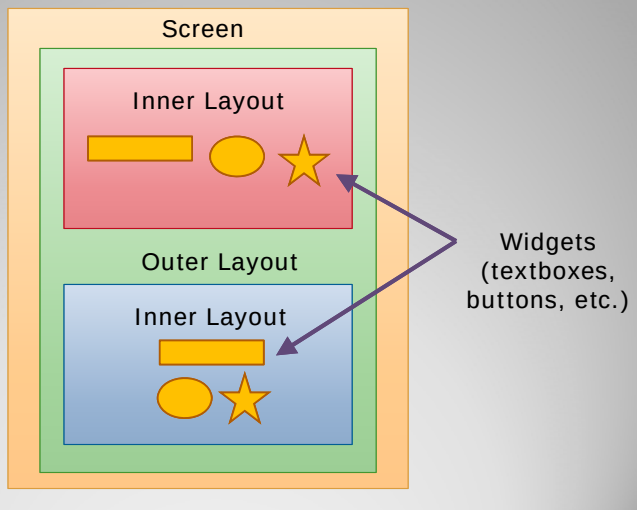
Layouts And Views

Android provides us with a number of pre-defined layouts and views to use for GUI design.

A Layout is a container. It will typically contain other layouts, and views.

A view is a graphical user element such as a button, textview, or textbox for input.

Layouts



Layouts and Views

As you've already seen, each layout and views object contains various properties, which can be defined in the XML layout file.

Some properties are common to most of the objects, while others are unique to a specific layout or view.

Example:

```
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content" android:id="@+id/txtOut"
android:textSize="18px" android:textStyle="bold"
android:layout_gravity="center_horizontal"
android:padding="5px" android:text="It's cloudy, so we're going
to class!">
</TextView>
```

Creating Properties

Properties can be added visually (using the Properties panel in Eclipse), or can be typed directly into the XML file. If using Eclipse when editing the file directly, Eclipse will provide hints to make your life easier.

The basic structure of any property is as follows:

android:property="value"

where property is one of the supported properties for the object (layout or widget), and value is one of the values supported by that property. In some cases, multiple values can be supplied, separated by a comma.

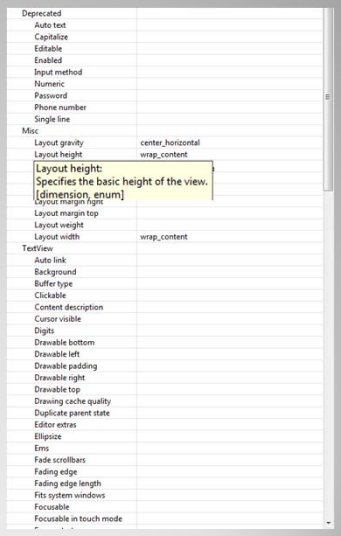
Example:

```
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content" android:id="@+id/txtOut"
android:textSize="18px" android:textStyle="bold"
android:layout_gravity="center_horizontal" android:padding="5px"
android:text="It's cloudy, so we're going to class!">
</TextView>
```

Getting Help

If you're not sure what a property does, use the hints Eclipse provides to give you a better understanding.

Even better, try setting the property and see what it does first hand.



Common Layout Properties

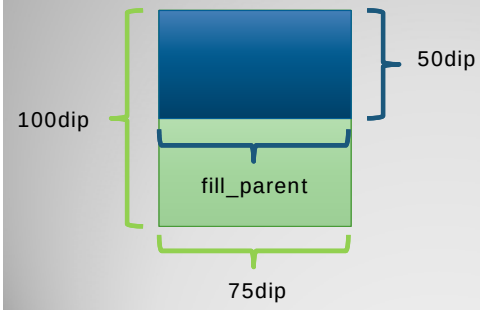
There are some common properties for most layout objects. They are:

- layout_height and layout_width
- layout_gravity
- layout_margin
- layout_weight (not as common)
- padding

layout_width and layout_height

Many layouts and views share the same properties. Layout_width and layout_height control the dimensions of each object.

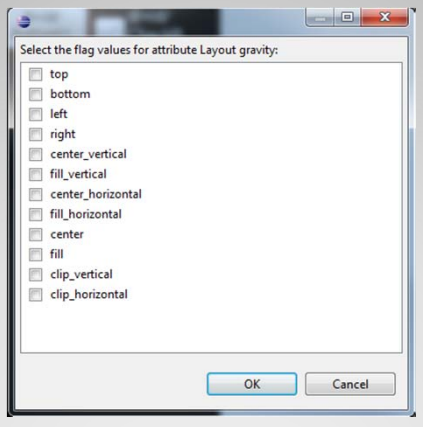
They can be set to be a specific pixel, or dip size, or they can be set to be the same size as their parent object by using the "fill_parent" value.



Be careful when working with multiple objects that use fill_parent. You may not be able to see or work with all of your objects, if each is set to fill the screen. The first one will be displayed, while the rest will be invisible to the users.

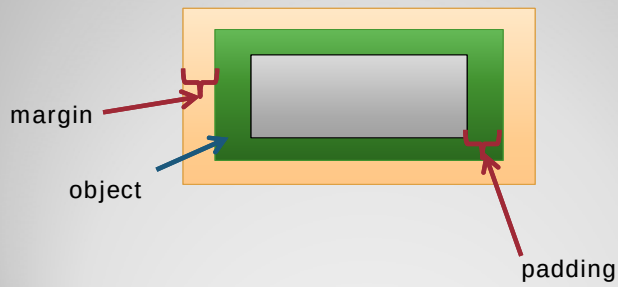
layout_gravity

Layout_gravity allows you to specify the orientation of an object.



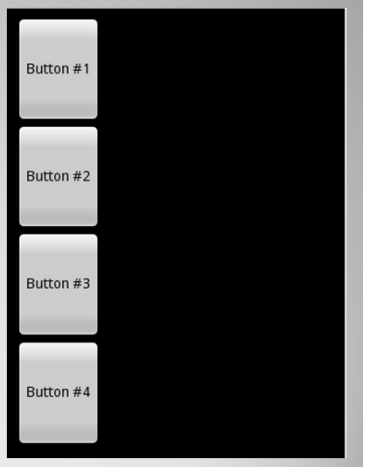
layout_margin and layout_padding

Layout_margin, and layout_marginLeft, layout_marginRight, layout_marginTop, and layout_marginBottom control the space outside of the object, while padding, paddingLeft, paddingRight, paddingTop, and paddingBottom control the inner space within an object.



layout_weight

Layout_weight is not as common, but is useful on occasion. It allows you to specify a ratio of how objects occupy the available space within a layout. For example, if you had two buttons, and you wanted to distribute them within your screen at 60% for the first button, and 40% for the second, you could specify a layout_weight of .6 for button one, and .4 for button two. Layout_weight across all objects needs to add up to 1. Here's an example of a layout where 4 buttons are set to .25 layout_weight each.



Views

Views are the elements that make up our GUI. The most common ones are:

- **TextView** – This is a label that displays a string of text on the screen.
- **EditText** – Can be single or multi-line. Allows users to input text, or numbers.
- **Button** – a clickable button that triggers an event when clicked.
- **ImageButton** – a button that users a graphic as its interface. Still produces a clickable event.
- **CheckBox** – allows the user to select any number of options out of a group.
- **RadioButton** and **RadioGroup**– allows the user to choose one option out of a group. A RadioGroup is defined first, and RadioButtons are defined with in it.

This is the list of views you'll use most frequently.

More Views

In addition to the basic elements covered on the previous slide, Android offers the following:

- **TimePicker** – allows the users to specify time.
- **DatePicker** – allows the users to specify a date.
- **ListView** – shows lists of items in a vertical scrolling list.
- **Spinner** – Similar to a drop-down list, a spinner allows a user to select one item out of a group.
- **Gallery** – Displays a gallery of images
- **ImageView** – displays a single image.
- **ImageSwitcher** – allows you to create a scrollable list of images, which when clicked display a larger version.
- **GridView** – displays a grid with rows and columns.
- **WebView** – allows you to display a web page within your application.
- **AnalogClock** – displays an analog clock.
- **DigitalClock** – displays a digital clock.

TextView

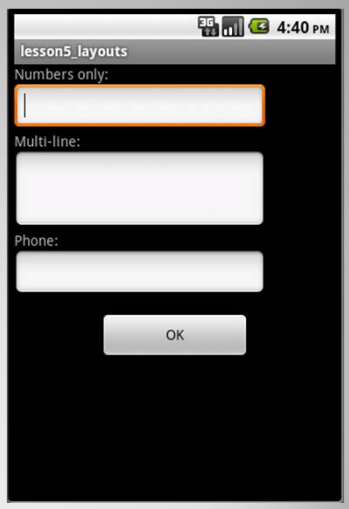
We're been using the TextView for quite some time now to output information out to the screen. It has a text property, which can be set from code using .setText() method, or can be referenced from a strings.xml file, or typed in directly, as in the example below:

```
<TextView android:text="This is my text!"
          android:id="@+id/TextView01"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"></TextView>
```

EditText

EditText view allows the user to type in information into the control. The control has an InputType property which can be used to specify what type of information is allowed (numeric, decimal or text), and how to work with it (capitalize the first letter, it's an name, URL, etc.). Depending on which mode is selected, a different keyboard will be provided to the user for input.

```
<EditText
android:id="@+id/EditText01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:inputType="number"
android:width="250dp">
</EditText>
```



EditText

From code, you can create an instance of an EditText object, and locate it's value by calling the getText() method.

Example:

```
final EditText tIn = (EditText) findViewById(R.id.EditText01);
String sMessage = tIn.getText().toString();
```

Because the information, even if numeric in nature, comes in as text, you will need to do a cast or a conversion on the resulting value before using it in calculations.

Button and ImageButton

A button allows you to produce a clickable event. Use the button's android:text property to set it's text value. Be sure to specify an id as well, as this will be necessary for the clickable event in your code.

In xml:

```
<Button android:layout_height="wrap_content"
android:id="@+id/btnOK" android:text="OK"
android:layout_width="150dp"
android:layout_gravity="center_horizontal"
android:layout_marginTop="15dp"></Button>
```

In code:

```
final Button btnOK = (Button) findViewById(R.id.btnOK);
    btnOK.setOnClickListener(new View.OnClickListener() {
           public void onClick(View v) {
              // Code for click goes here
           }
       });
```

Checkbox

A checkbox allows you to specify one or more options that can be selected by a user.

In XML:

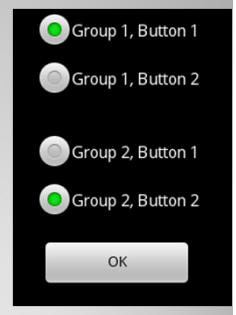
```
<CheckBox android:id="@+id/CheckBox01"
android:layout_width="wrap_content"
android:layout_height="wrap_content" android:text="Checked!"
android:checked="true"></CheckBox>
```

In Code:

```
final CheckBox oCheck =(CheckBox) findViewById(R.id.CheckBox01);
    if (oCheck.isChecked()) {
        oCheck.setText("Checked!");
    }
    else {
        oCheck.setText("Not Checked!");
    }
```

RadioButton & RadioGroup

A RadioGroup defines a set of RadioButtons. Within each group, only ONE RadioButton can be selected. Assuming you have multiple groups on the same screen, a user is able to select one radiobutton from each of the groups.

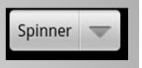


XML:

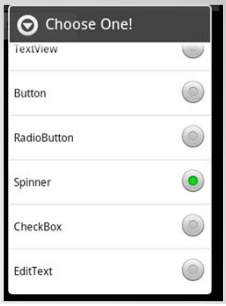
```
<RadioGroup android:id="@+id/RadioGroup01"
android:layout_width="wrap_content"
android:layout_height="wrap_content">

  <RadioButton android:id="@+id/RadioButton01"
android:layout_width="wrap_content"
android:checked="true"
android:layout_height="wrap_content"
android:text="Group 1, Button 1"></RadioButton>
  <RadioButton android:id="@+id/RadioButton02"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Group 1, Button 2"></RadioButton>

</RadioGroup>
```

Spinner

A spinner is a drop-down list of choices, which is very similar to a group of radio buttons, in the sense that only one option out of several can be selected. However, the spinner doesn't take up a lot of room on the screen, so it will allow for more options for the user to choose from.




```
<Spinner android:id="@+id/Spinner01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:prompt="@string/spinheader">
</Spinner>
```



Spinner

To load a spinner with data, we'll typically need some sort of array, or data table. We will also need an appropriate Adapter, which is a pre-programmed object that helps us manage the data source.

```
String[] items = new String[] {
"TextView", "Button", "RadioButton",
"Spinner", "CheckBox", "EditText", "WebView"};

Spinner spinner = (Spinner) findViewById(R.id.Spinner01);
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_spinner_item, items);
adapter.setDropDownViewResource(
          android.R.layout.simple_spinner_dropdown_item);

spinner.setAdapter(adapter);
```

Spinner

To figure out what the user selected, we need to create a child class that implements OnItemSelectedListener, then set the spinner's OnItemSelectedLister to the instance of that class.

Example:

```
public class MyOnItemSelectedListener implements OnItemSelectedListener {
 public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2,long arg3) {
  Spinner spinner = (Spinner) findViewById(R.id.Spinner01);
  String sSelected = spinner.getSelectedItem().toString();
 }

 public void onNothingSelected(AdapterView<?> arg0) {
 }
}
```

And in our main event (onCreate):

```
spinner.setOnItemSelectedListener(new MyOnItemSelectedListener());
```

To set a value on a spinner, use:

```
spinner.setSelection(value);
```

Value is an integer, representing one of the items in the list (0 based).

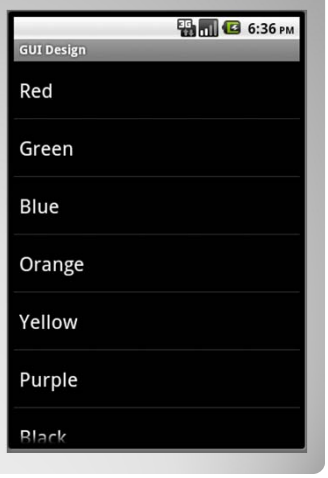
ListView

A ListView provides a scrollable list of items. This is similar to a Spinner, where one item can be selected.

```
<ListView
android:id="@+id/android:list"
android:layout_width="wrap_content"
android:layout_height="wrap_content">
</ListView>
```

Setting and retrieving information from a ListView is similar to how we did it with the Spinner control.

In order to avoid exceptions, you must use the id **@+id/android:list** in your XML.



ListView

```
public class main_activity extends ListActivity {

String[] viewlist = {
            "Red","Green","Blue","Orange","Yellow","Purple","Black",
            "White","Cyan","Magenta","Fucia","Lime","Baige","Gray"
    };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        setListAdapter(new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, viewlist));
    }

    public void onListItemClick(
        ListView parent, View v,
        int position, long id)
        {
            //position contains the index of the selected item
            //viewlist[position] = text of the selected item
        }
}
```

WebView

A WebView allows you to display a web page within your application. The page can be a local html file, located in the assets folder, or it could be a live web page located online.

XML:

```
<WebView android:id="@+id/webview01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"/>
```

WebView

The most basic implementation of a webview is below:

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

        final WebView oWeb = (WebView) findViewById(R.id.webview01);
        oWeb.getSettings().setJavaScriptEnabled(true);
        oWeb.loadUrl("http://www.google.com");
        oWeb.setWebViewClient(new oWebViewClient());

  }
//necessary for keeping clicked links within your webview, otherwise,
//android will open a new window outside of your app.
  private class oWebViewClient extends WebViewClient {
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
        view.loadUrl(url);
        return true;
    }
}
```

Layouts

Layouts are the containers for your views. You've already seen the LinearLayout in action, and here are a few more popular choices:

- **ScrollView** – Allows you to create a single column, much like the LinearLayout, except that scrollbars are available. This layout type is useful when you have a long list of options for the user to work with.
- **TableLayout** – This layout formats your content in rows / cells, much like an HTML table. Unfortunately, it doesn't provide a lot of control over how the table looks, and what size is applied to each cell. Cells are sized by the largest element within a column. Borders for TableLayout are not visible.

Layouts

Class	Description
FrameLayout	Layout that acts as a view frame to display a single object.
Gallery	A horizontal scrolling display of images, from a bound list.
GridView	Displays a scrolling grid of m columns and n rows.
LinearLayout	A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.
ListView	Displays a scrolling single column list.
RelativeLayout	Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).
ScrollView	A vertically scrolling column of elements.
SurfaceView	Provides direct access to a dedicated drawing surface. It can hold child views layered on top of the surface, but is intended for applications that need to draw pixels, rather than using widgets.
TabHost	Provides a tab selection list that monitors clicks and enables the application to change the screen whenever a tab is clicked.
TableLayout	A tabular layout with an arbitrary number of rows and columns, each cell holding the widget of your choice. The rows resize to fit the largest column. The cell borders are not visible.
ViewFlipper	A list that displays one item at a time, inside a one-row textbox. It can be set to swap items at timed intervals, like a slide show.
ViewSwitcher	Same as ViewFlipper.

http://developer.android.com/guide/topics/ui/layout-objects.html

LinearLayout

In a LinearLayout, all child objects are arranged either in horizontal or vertical fashion within a single column on the screen.

The following property controls LinearLayout orientation:

```
android:orientation="vertical" or
android:orientation="horizontal"

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">

……

</LinearLayout>
```

ScrollView

A ScrollView will typically be used in conjunction with a second layout. For example, you can create a ScrollView, and place a LinearLayout or a TableLayout inside the ScrollView. ScrollView provides properties for Scrollbars (horizontal, and vertical), and their size and style.

```
<ScrollView android:id="@+id/ScrollView01"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
xmlns:android="http://schemas.android.com/apk/res/android"
android:scrollbars="horizontal|vertical">

……

</ ScrollView >
```

TableLayout and TableRow

A TableLayout allows you to display your widgets inside a grid. The number or rows in the grid is determined by the <TableRow> tag, while the number of cells is determined by the number of views you add to each row.

```
<TableLayout android:id="@+id/TableLayout01"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
xmlns:android="http://schemas.android.com/apk/res/android">
<TableRow android:id="@+id/TableRow01"
android:layout_width="wrap_content"
android:layout_height="wrap_content>
…
</TableRow>
…
</TableLayout>
```