ECE155: Engineering Design with Embedded Systems Winter 2013

Lecture 22 — March 8, 2013

Patrick Lam *version 1*

The first half of this lecture was a brief summary of what you need to do for Lab 4. Since we have a detailed lab handout, I won't go over it in the PDF notes.

Two Random Android Nuggets

Before going on to the main topic of today's lecture, I'll discuss two nuggets: when to use XML versus Java code for adding widgets to your layouts; and what "inflate" means in the code that you always auto-generate.

XML versus Java. Use the right tool for the job! Sometimes this is XML; other times, it's Java. Here's the tradeoff.

XML = more safety:

- You can select and place items on your layouts at any point.
- You don't need to run your code under the emulator (or on a phone) to see how things will look.
- You get more error checking: the compiler can tell you about what you're doing wrong and help you fix it ahead of time.

Java Code = more flexibility:

- You can choose widgets based on user input or your computations.
- You can use loops or other control flow to generate related items.
- You get less error checking of what you've written.

What "inflate" means. These lines keep on showing up in our auto-generated `Activity` code:

```
// Inflate the menu; this adds items to the action bar if it is present.
getMenuInflater().inflate(R.menu.activity_main, menu);
```

"Inflate" means taking an XML tree and creating a tree of `View` objects, based on the description in the XML.

Android Graphics

Next, we'll see how to put graphics on the screen in Android. There are, in general, two options:

- use a `View` (easier; but suitable only for infrequent updates); or
- paint to a `Canvas` (more complicated; but permits real-time updates).

Even though the `Canvas` is more suitable for embedded systems, the `View` is easier, so we'll be talking about it in this class.

Drawing to a `View`.

As always with Android, you can put things onto the `View` either by specifying XML, or programmatically. In either case, you will use a `Drawable`, either explicitly or implicitly.

The `Drawable` class. A `Drawable` object represents, as its name suggests, "something that can be drawn". Examples include:

- BitmapDrawable: draw a bitmap onto the screen;
- ShapeDrawable: draw a shape;
- PictureDrawable: play back a sequence of drawing calls.
- etc.

All of the drawing techniques we're going to see are going to create a `Drawable` of some sort.

Drawing Bitmaps. The easiest way to get some graphics onto your screen is by drawing them in some other program and then including them:

- put a picture (`PNG`, `JPG` or `GIF`) in `res/drawables`; and
- use an `ImageView` to include it on the screen.

Examples from: `http://developer.android.com/guide/topics/graphics/2d-graphics.html`

```
// Instantiate an ImageView and define its properties programmatically
ImageView i = new ImageView(this);
i.setImageResource(R.drawable.my_image);
 // set the ImageView bounds to match the Drawable's dimensions
i.setAdjustViewBounds(true);
i.setLayoutParams(new Gallery.LayoutParams
    (LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));

// Add the ImageView to the layout and
//  set the layout as the content view
mLinearLayout.addView(i);
```

Or, you can do it through XML. Again, you need the appropriate drawable in the `res/drawables` directory. Note that since we're using XML, it's harder to go wrong.

```
<ImageView
  android:id="@+id/imageView1"
  android:layout_height="wrap_content"
  android:layout_width="wrap_content"
  android:src="@drawable/myImage" />
```

`ShapeDrawable`. Next, we'll see how to draw our own shapes. Primitive shapes include:

- PathShape—lines;
- RectShape—rectangles;
- OvalShape—ovals and rings;

Once again, we put these into an `ImageView`.

You can draw shapes in XML and put them into an `ImageView`. Again, you need the appropriate drawable in the `res/drawables` directory. Add this snippet, which creates the `ImageView` backed by the drawable to your Layout XML:

```
<ImageView android:id="@+id/imageView2"
  android:src="@drawable/cyan_shape" ... />
```

Next, create a separate XML file for the drawable itself. This actually provides instructions for Android to put dots on the screen.

```
<shape android:shape="oval" ... >
  <size android:width="160px" android:height="160px" />
  <solid android:color="#7f00ffff" />
</shape>
```

Everything you can do in XML, you can do in code. Here, we specify the shape in Java.

```
private class MyDrawableView extends ImageView {
  private ShapeDrawable mDrawable;
  public MyDrawableView(Context context, int color) {
    ...
    mDrawable = new ShapeDrawable(new OvalShape());
    mDrawable.getPaint().setColor(color);
    mDrawable.setBounds(0, 0, size, size);
    mDrawable.setAlpha(alpha);
  }
  protected void onDraw(Canvas canvas) {
    mDrawable.draw(canvas);
  }
}
```

Then, in the Activity's `onCreate()`, we would instantiate this `MyDrawableView` and add it to the parent view:

```
MyDrawableView magentaView =
  new MyDrawableView(this, Color.MAGENTA);
magentaView.setLayoutParams
  (new LinearLayout.LayoutParams(160, 160));
addView(magentaView);
```