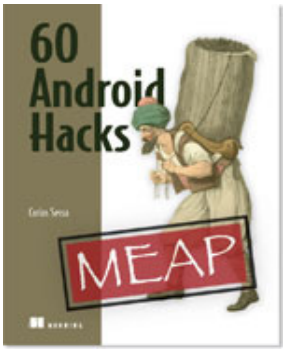
MANNING PUBLICATIONS



60 Android Hacks
By Carlos Sessa

To draw directly on the screen, you can use Android's Canvas class. In this hack from 60 Android Hacks, the author shows how to use this class to create a box that bounces around screen.

Making Animations over the Canvas

If you're animating your own widgets, you might find that the Android APIs are a bit limited. Is there an Android API to draw things directly to the screen? The answer is yes. Android offers us a class called `Canvas`. In this hack, we see how we can use the `Canvas` class to draw elements and animate them by creating a box that will bounce around screen.

First of all, let's make sure we understand what the `Canvas` class is. A `Canvas` works for you as a pretense, or interface, to the actual surface upon which your graphics will be drawn—it holds all of your "draw" calls. Via the `Canvas`, your drawing is actually performed upon an underlying `Bitmap`, which is placed into the window.

So the `Canvas` class holds all the `draw` calls. We can create a `View`, override the `onDraw()` method, and start drawing primitives there. To make everything clearer, we'll crate a `DrawView` class that will take care of drawing the box and updating its position. Since we don't have anything else onscreen, we'll make it the activity's content view. Here's the code for the `Activity`:

```
public class MainActivity extends Activity {

  private DrawView mDrawView;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Display display = getWindowManager().getDefaultDisplay();          #1
    mDrawView = new DrawView(this);
    mDrawView.height = display.getHeight();
    mDrawView.width = display.getWidth();

    setContentView(mDrawView);                                         #2
  }
}
```

In #1, we use the `WindowManager` to get the screen width and height. These values will be used inside the `DrawView` to limit where to draw. In we set the `DrawView` as the `Activity`'s `contentView`. This means that the `DrawView` will take all the available space.

Let's take a look of what's done inside the `DrawView` class:

```
public class DrawView extends View {
  private Rectangle mRectangle;
  public int width;
  public int height;
```

```
    public DrawView(Context context) {
      super(context);

      mRectangle = new Rectangle(context, this);                    #1
      mRectangle.setARGB(255, 255, 0, 0);
      mRectangle.setSpeedX(3);
      mRectangle.setSpeedY(3);
    }

    @Override
    protected void onDraw(Canvas canvas) {                          #2
      mRectangle.move();                                            #3
      mRectangle.onDraw(canvas);                                    #4

      invalidate();
    }
  }
```

In #1, we create a `Rectangle` instance that will play the role of the box. The `Rectangle` class also knows how to draw itself to a canvas and contains all the boring logic of how to update its position to get drawn in the correct place. When the `onDraw()` method is called, we change the rectangle's position (#2) and we draw it to the canvas (#3). The `invalidate()` call in #4 is the hack itself.

The `invalidate()` is a `View`'s method to force a view to draw. Placing it inside the `onDraw()` method means that `onDraw()` will be called as soon as the view finishes drawing itself. To put it differently, we'll loop over the `Rectangle` 's `move()` and `onDraw()` calls, creating a nice animation!

Summary

Updating views positions in the `onDraw()` method through the `invalidate()` call is an easy way to provide custom animations. If you're planning to make a small game, using this trick is a simple way to handle the game's main loop.

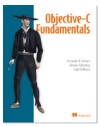
External links

http://developer.android.com/reference/android/graphics/Canvas.html
http://developer.android.com/guide/topics/graphics/2d-graphics.html

Here are some other Manning titles you might be interested in:

Android in Action, Third Edition
W. Frank Ableson, Robi Sen, Chris King, and C. Enrique Ortiz

Android in Practice
Charlie Collins, Michael D. Galpin, and Matthias Kaeppler



Objective-C Fundamentals
Christopher K. Fairbairn, Johannes Fahrenkrug, and Collin Ruffenach

Last updated: March 16, 2012