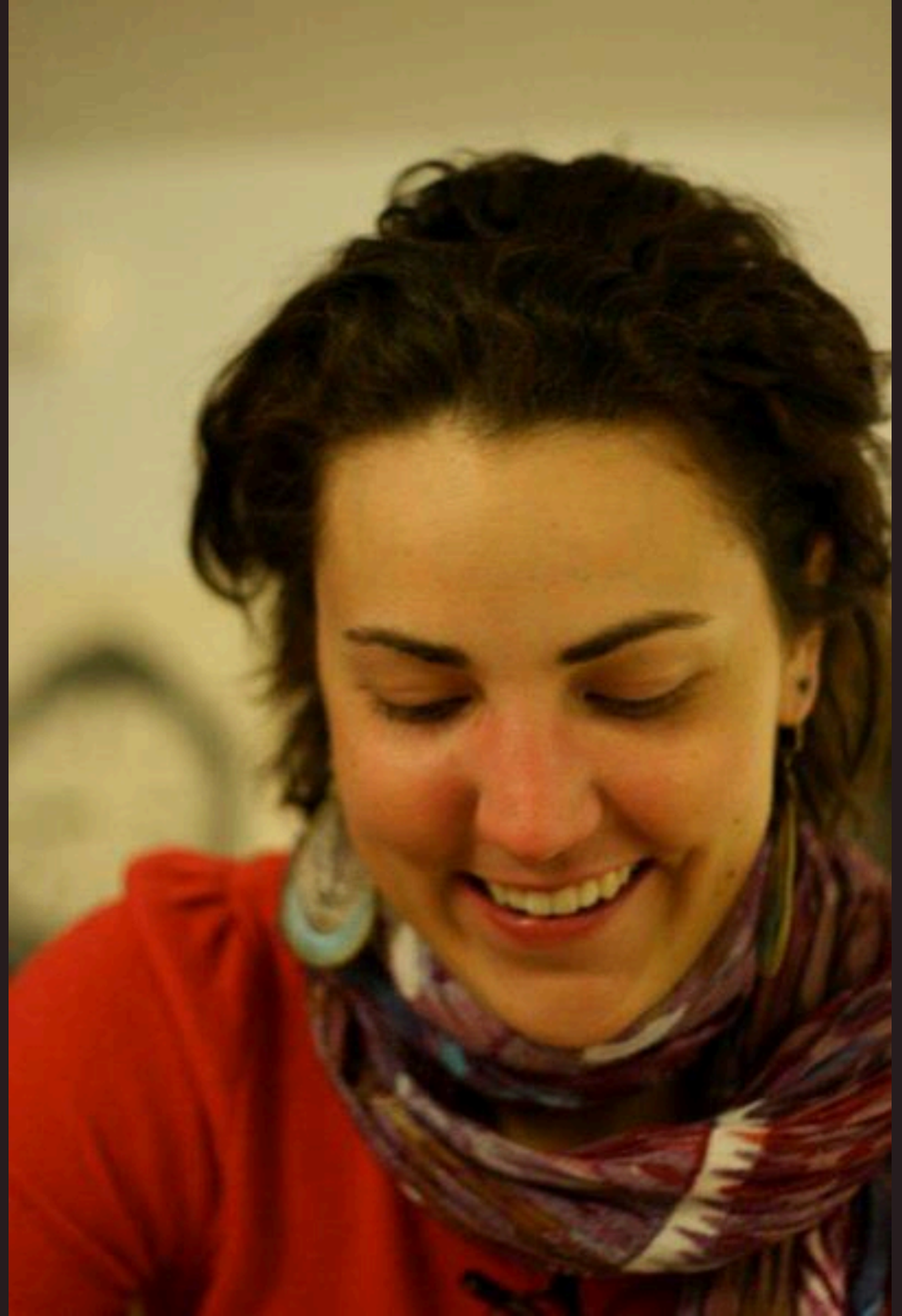
getBounds()

The story of Drawables and their View masters

Jamie Huson + Lisa Neigut | 20 Sept 2014



Drawables!

What is a Drawable?

"A drawable is a positioned entity

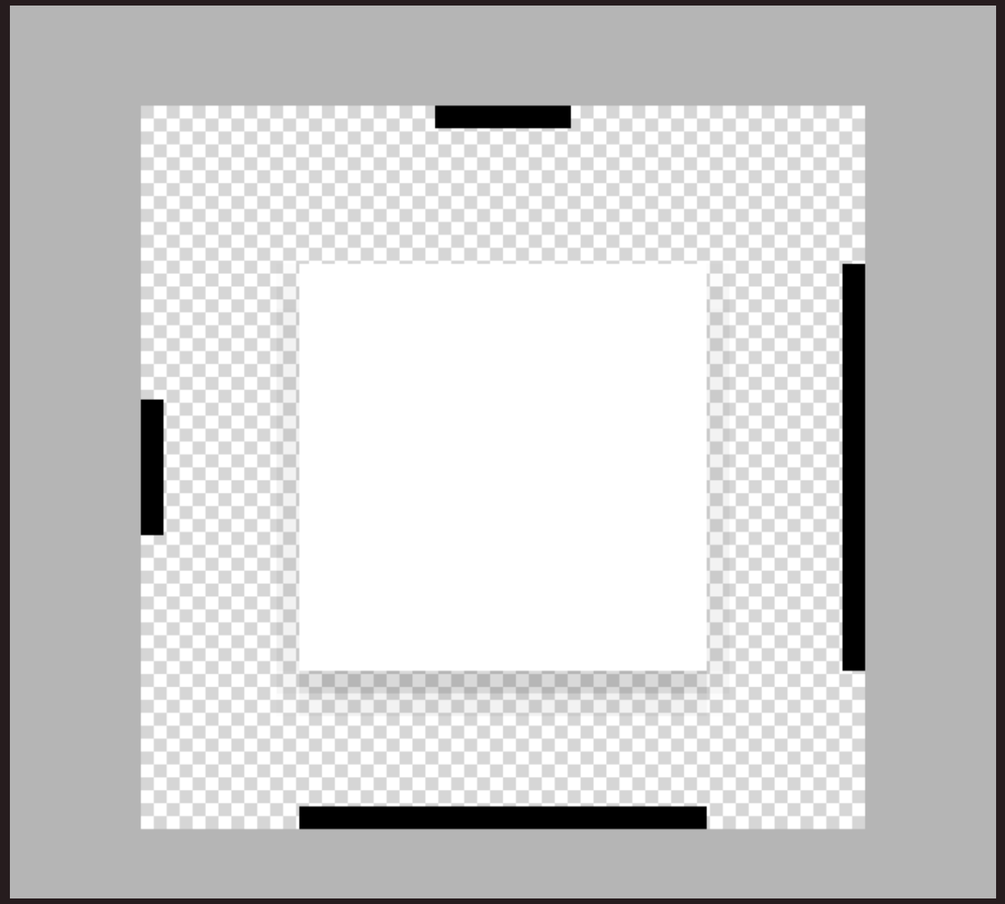
that is to be drawn on a canvas."

-Cyril Mottier

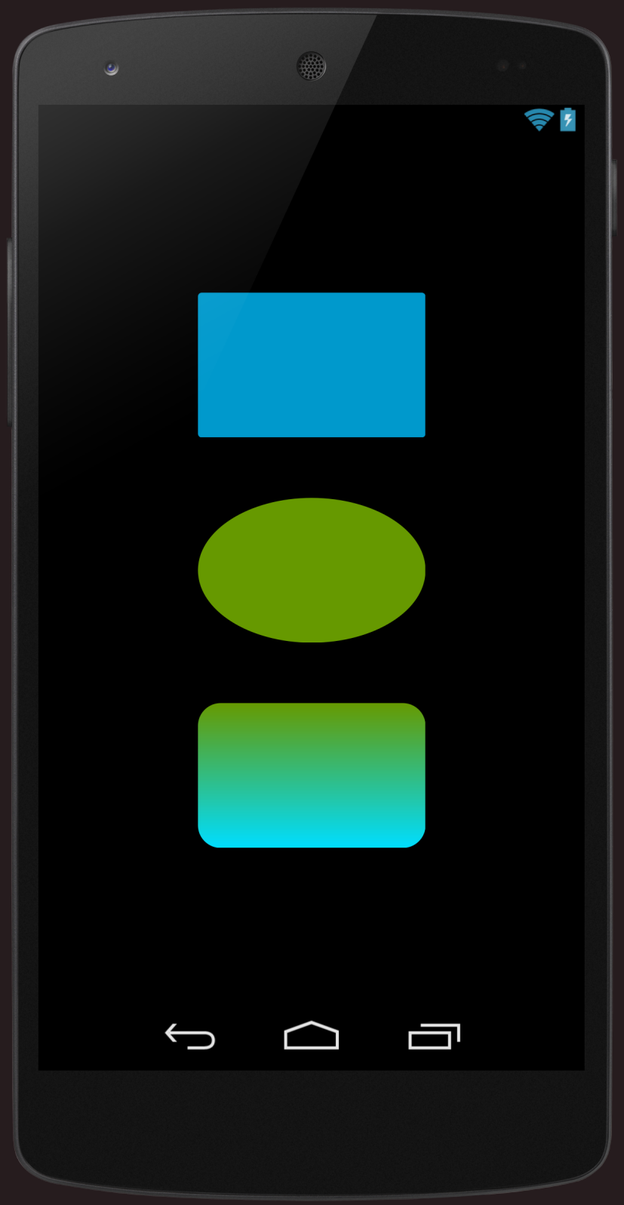
What Makes Drawables Awesome:

- Drawing is fun!
- State Changes
- Total Separation of View Logic from Code
- Focused code
- XML

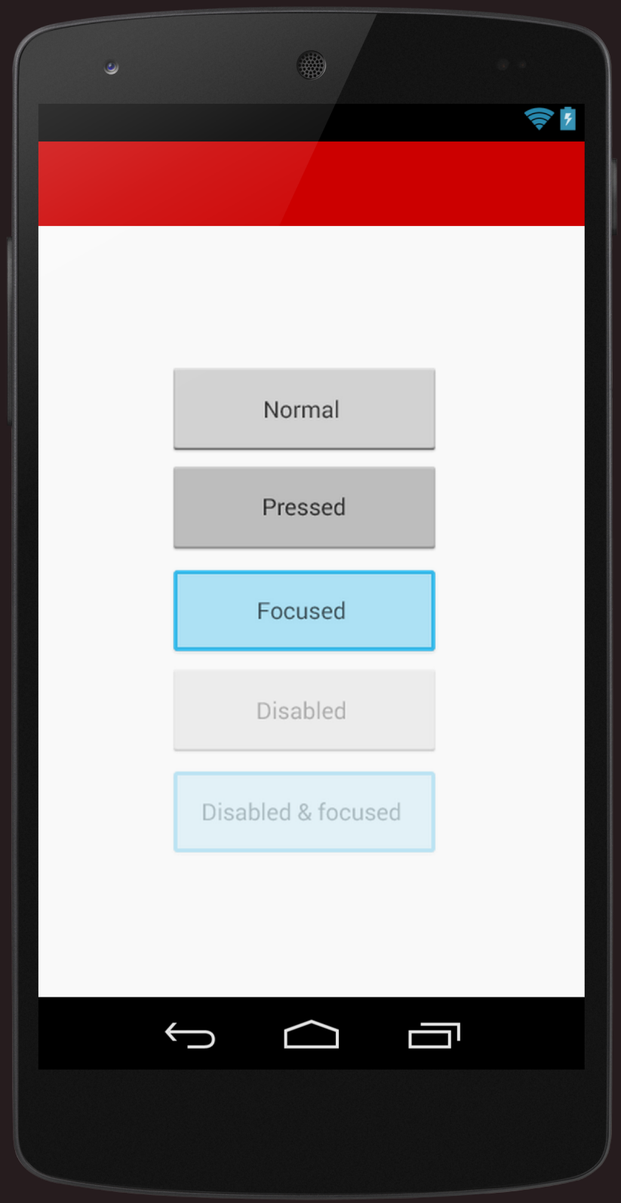
NinePatchDrawable



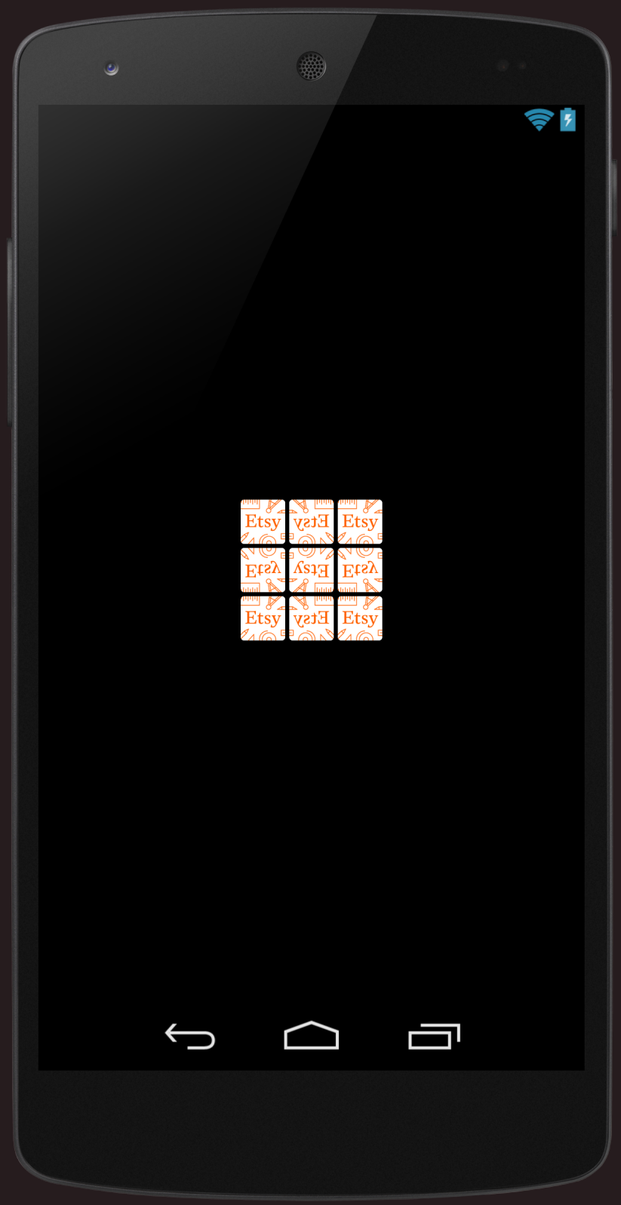
ShapeDrawable



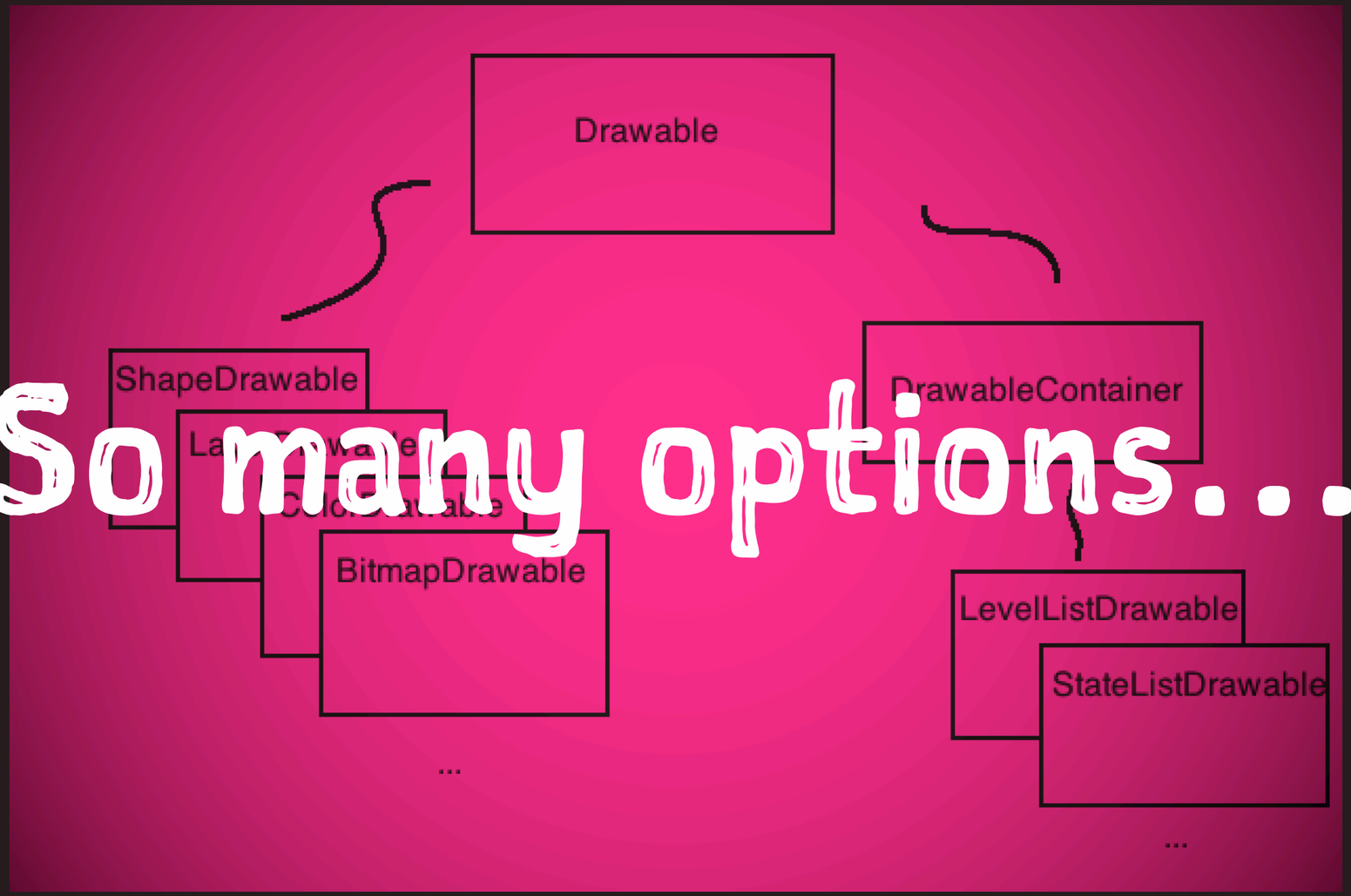
StateListDrawable



BitmapDrawable

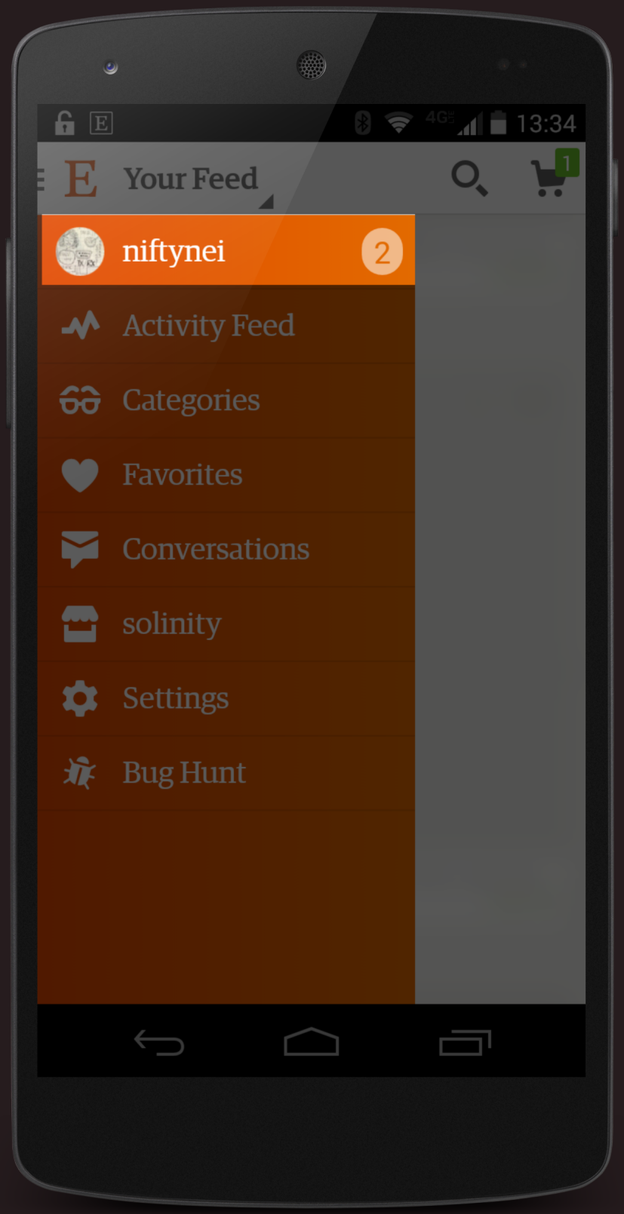


LayerDrawable

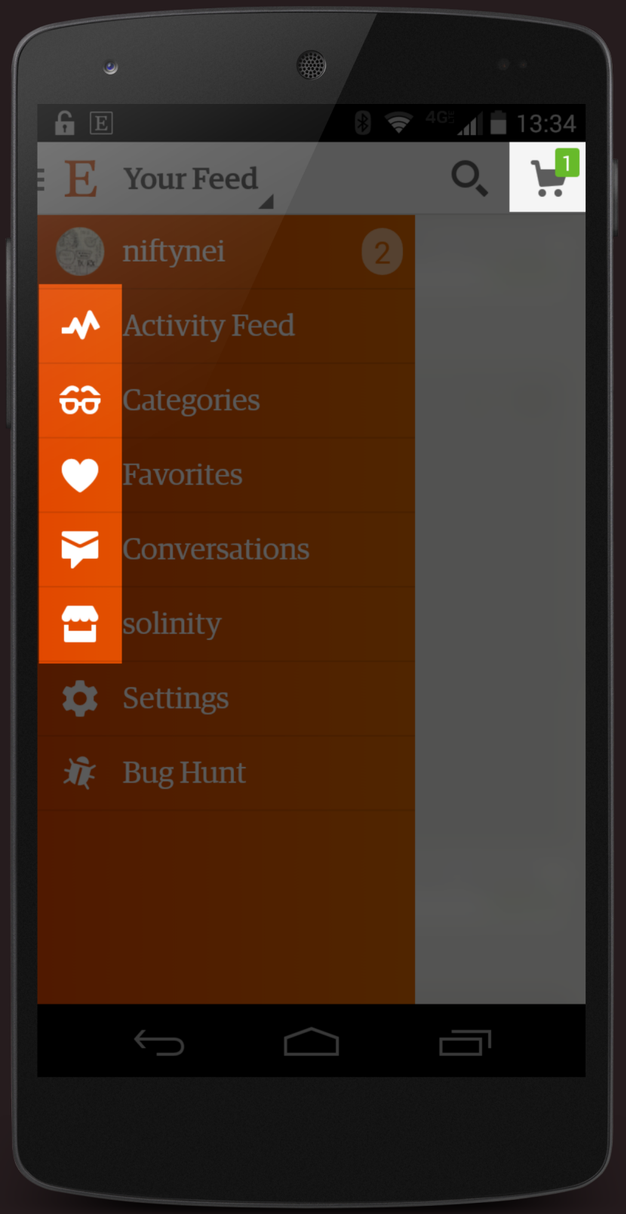


So many options...

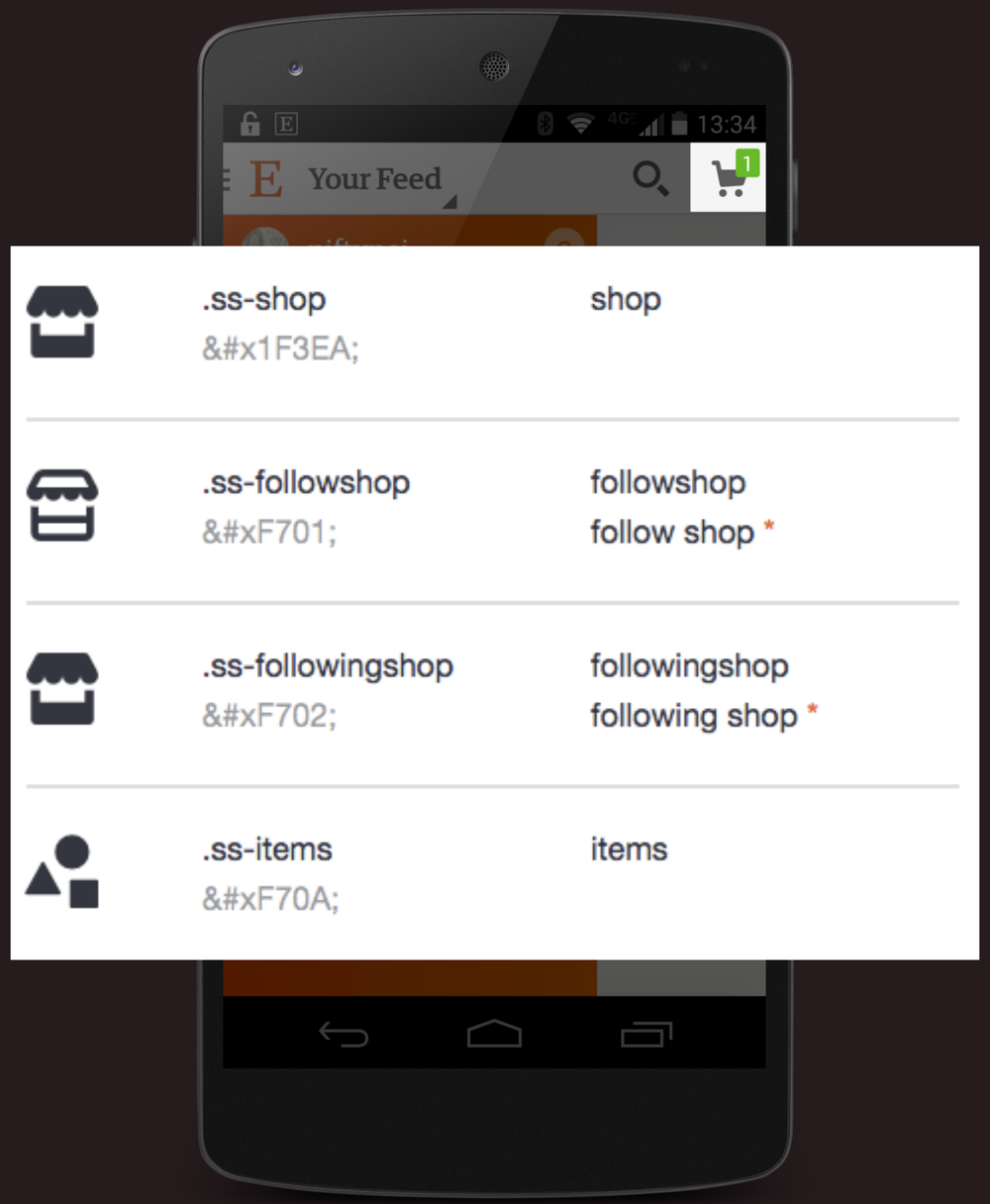
BadgeDrawable



IconDrawable



IconDrawable



Custom Drawables

Using built-in Drawables

/res

../drawable

.. layered_drawable.xml

```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Transparent top line for evenness -->
    <item>
        <shape android:shape="oval">
            <solid android:color="@color/transparent"/>
            <corners android:radius="@dimen/gen_avatar_corners_small"/>
            <padding
                android:top="@dimen/gen_avatar_border_shadow"/>
        </shape>
    </item>
    <!-- Light grey -->
    ...
```

Using built-in Drawables

layered_drawable.xml

```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Transparent top line for evenness -->
    <item>
        <shape android:shape="oval">
            <solid android:color="@color/transparent"/>
            <corners android:radius="@dimen/gen_avatar_corners_small"/>
            <padding
                android:top="@dimen/gen_avatar_border_shadow"/>
        </shape>
    </item>
    <!-- Light grey -->
    ...
```

layout.xml

```
<View
    ...
    android:background="@drawable/layered_drawable"
/>
```

Custom Drawable

/res

../drawable

..custom_drawable.xml

```
<custom-drawable xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    app:icon="@id/ic_etsy_e"
    app:color="@color/etsy_orange" >
</custom-drawable>
```

Custom Drawable

custom_drawable.xml

```
<custom-drawable xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    app:icon="@id/ic_etsy_e"
    app:color="@color/etsy_orange" >
</custom-drawable>
```

layout.xml

```
<View
    ...
    android:background="@drawable/custom_drawable”
/>
```

Runtime Error

```
Caused by: org.xmlpull.v1.XmlPullParserException:
    Binary XML file line #2: __invalid drawable tag custom__
```

Drawable.java

```
public static Drawable createFromXmlInner(Resources r, XmlPullParser parser, AttributeSet attrs)
throws XmlPullParserException, IOException {
    Drawable drawable;

    final String name = parser.getName();

    if (name.equals("selector")) {
        drawable = new StateListDrawable();
    } else if (name.equals("level-list")) {
        drawable = new LevelListDrawable();
    } else if (name.equals("layer-list")) {
        drawable = new LayerDrawable();
    } else if (name.equals("transition")) {
        drawable = new TransitionDrawable();
    } else if (name.equals("color")) {
        drawable = new ColorDrawable();
    } else if (name.equals("shape")) {
        drawable = new GradientDrawable();
    }
    ...
    } else {
        throw new XmlPullParserException(parser.getPositionDescription() +
        ": invalid drawable tag " + name);
    }

    drawable.inflate(r, parser, attrs);
    return drawable;
}
```

Seriously?


made with duuigii.appspot.com

Good ol' Java

```
CustomDrawable drawable = new CustomDrawable();
drawable.setColorFilter(Color.BLACK, PorterDuff.Mode.DST);
mView.setBackground(drawable);
```

The Drawable API

Methods to @Override:

- draw(Canvas canvas)
- getOpacity()
- getIntrinsicHeight/Width()
- getMinimumHeight/Width()
- getConstantState() and mutate()

```
@Override
draw(Canvas canvas)
```

```
@Override
draw(Canvas canvas)
```

- Just like a View's drawing.
- Call `canvas.drawSomething(Paint)` methods.
- Transformations, Rotations, Shaders all still apply.
- Use `getBounds()` to determine drawing area

```
@Override
getOpacity()
```

@Override
getOpacity()

ColorDrawable.java

```
public int getOpacity() {
    switch (mState.mUseColor >>> 24) {
        case 255:
            return PixelFormat.OPAQUE;
        case 0:
            return PixelFormat.TRANSPARENT;
    }
    return PixelFormat.TRANSLUCENT;
}
```

```
@Override
getIntrinsicHeight\Width()
```

```
@Override
getIntrinsicHeight\Width()
```

ColorDrawable -> Drawable

```
    public int getIntrinsicHeight() {
        return -1;
    }
```

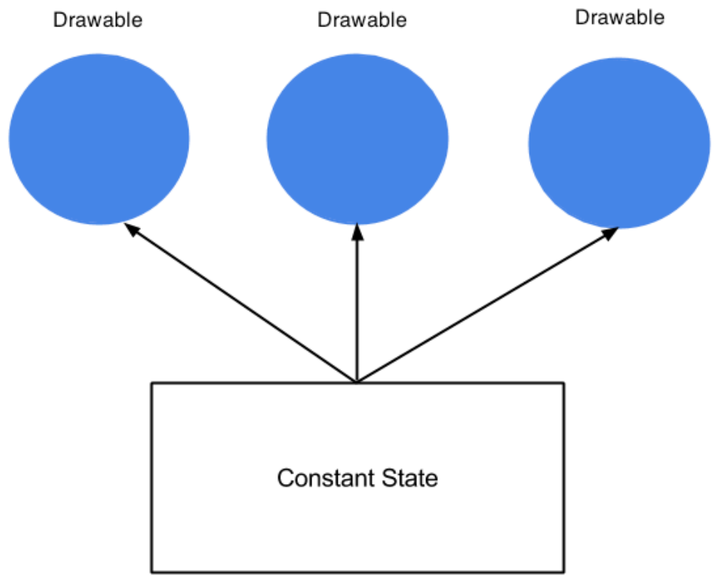
```
@Override
getMinimumHeight\Width()
```

```
@Override
getMinimumHeight\Width()
```

Default: returns 0

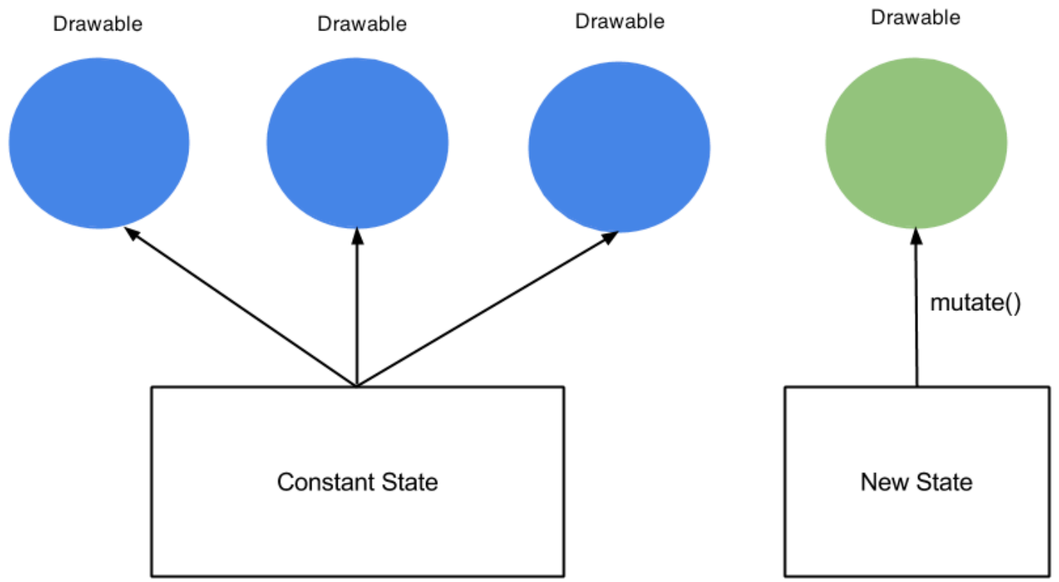
@Override
getConstantState() and mutate()

```
@Override
getConstantState()
```



```
@Override
mutate()
```

- Make this drawable mutable, independent of other instances.
- Basically creates an 'independent' state.



Drawables + Views

Drawable.Callback

Drawable.Callback

```
public static interface Callback {
    public void invalidateDrawable(Drawable who);
    public void scheduleDrawable(Drawable who, Runnable what, long when);
    public void unscheduleDrawable(Drawable who, Runnable what);
}
```

Drawable.Callback

```
public class View implements Drawable.Callback {

    ...

    public void setBackground(Drawable d) {
        mBackground = d;
        d.setCallback(this);
    }

    ...

    public void invalidateDrawable(Drawable drawable) {
        ...
            final Rect dirty = drawable.getBounds();
            final int scrollX = mScrollX;
            final int scrollY = mScrollY;
            invalidate(dirty.left + scrollX, dirty.top + scrollY,
                    dirty.right + scrollX, dirty.bottom + scrollY);
        ...
    }

    ...

}
```

Drawable.Callback

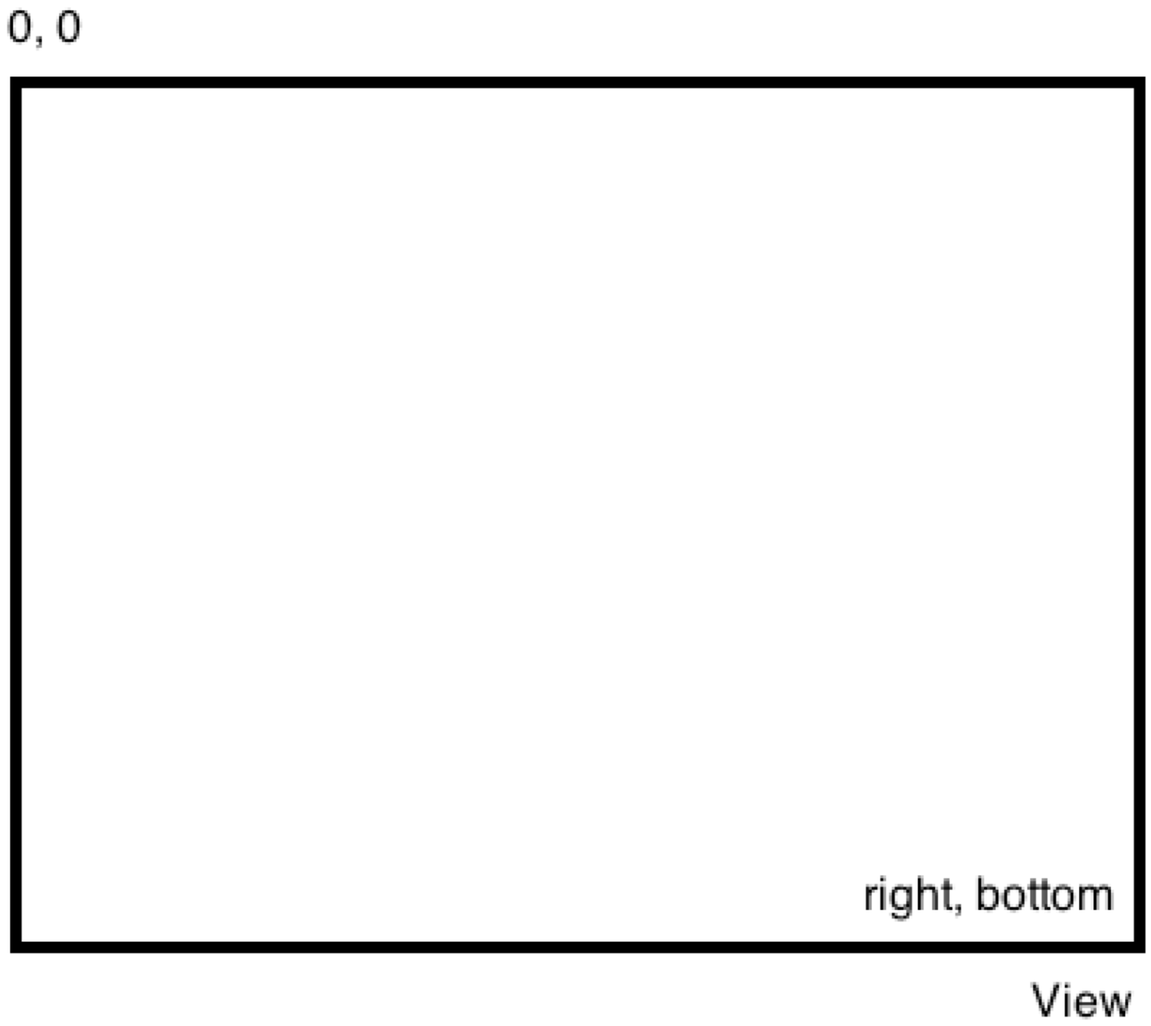
```
public abstract class Drawable {
    private WeakReference<Callback> mCallback = null;

    ...

    public void invalidateSelf() {
        final Callback callback = getCallback();
        if (callback != null) {
            callback.invalidateDrawable(this);
        }
    }

    ...

}
```

Drawable Bounds

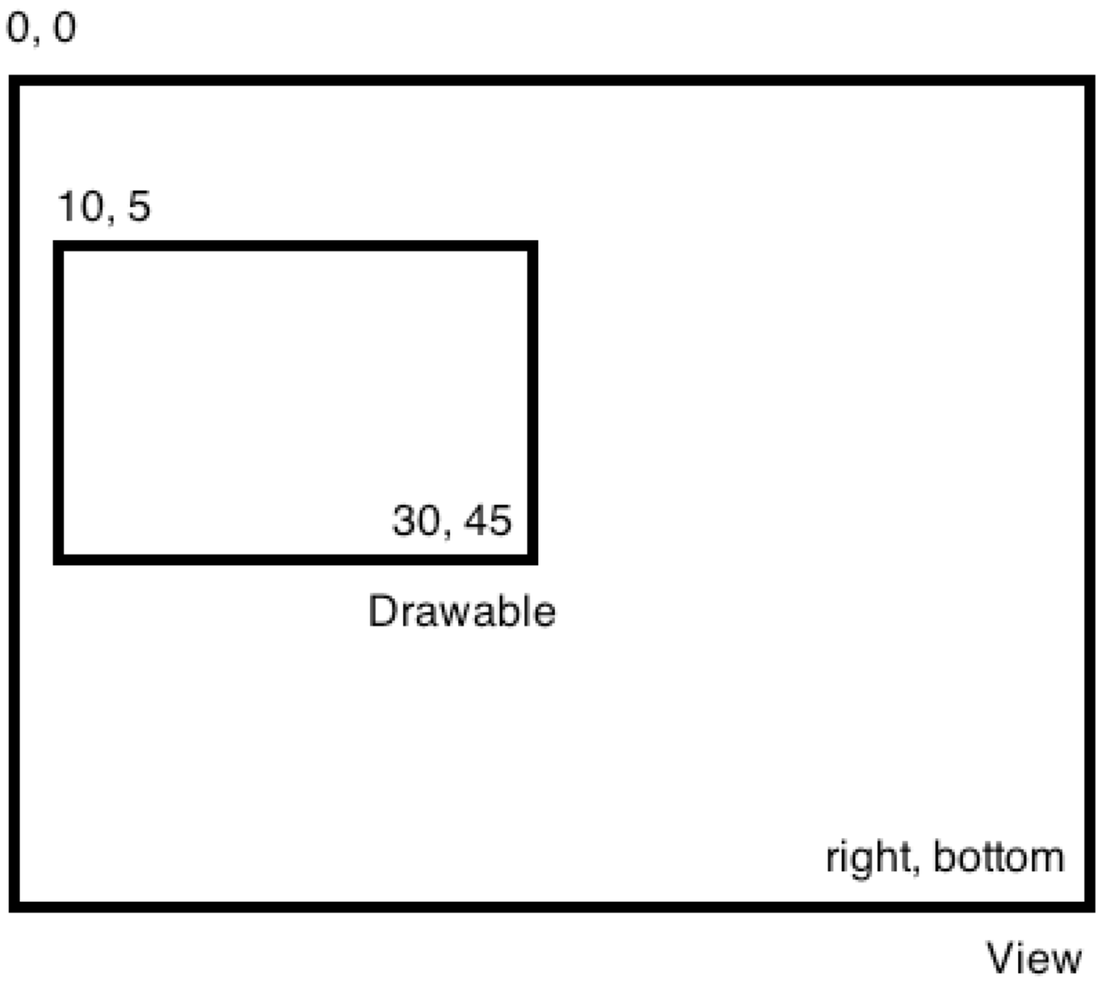
Drawable.java

```
setBounds(Rect rect)
setBounds(int left, int top, int right, int bottom)
```

```
setBounds(Rect rect)
setBounds(int left, int top,
int right, int bottom)
```



```
setBounds(Rect rect)
setBounds(int left, int top,
int right, int bottom)
```



View Render Passes

- Measurement
- Layout
- Draw

View Render Passes

- Measurement
- Layout
- Draw

onMeasure

- View queries the Drawable for its desired size

  `getIntrinsic…()`

onMeasure

- View queries the Drawable for its desired size
  `getIntrinsic…()`
- View uses the dimens of the Drawable to calculate its dimens and reports back its size to its parent

onDraw

→ Calculates the bounds for that drawable*

onDraw

- Calculates the bounds for that drawable*
- Calls `setBounds()` on the drawable*

onDraw

- Calculates the bounds for that drawable*
- Calls `setBounds()` on the drawable*
- Calls `draw(canvas)` on the drawable

A Few Notes:

Views assume that the drawable knows what size it wants to be before the bounds get set.

A Few Notes:

Views assume that the drawable knows what size it wants to be before the bounds get set.

Every View computes the drawable's bounds differently.

View Bound Calculations:

ColorDrawable.java

```
getIntrinsicHeight() { returns -1; }
```

View Bound Calculations:

ColorDrawable.java

```
getIntrinsicHeight() { returns -1; }
```

CompoundButton

```
setBounds(-1, -1, -1, -1);
```

View Bound Calculations:

ColorDrawable.java

```
getIntrinsicHeight() { returns -1; }
```

CompoundButton

```
setBounds(-1, -1, -1, -1);
```

ImageView

```
setBounds(leftMax, rightMax, topMax, bottomMax)
```

The Future of Drawable

Ripple

Ripple

```
public class Drawable {
    ...

    public void setHotspot(float x, float y) { /* compiled code */ }

    ...
}
```

Pre-L?

RippleDrawableCompat

```
public class RippleDrawableCompat extends LayerDrawable {

}
```

RippleDrawableCompat

```
public class RippleDrawableCompat extends LayerDrawable {

    private Drawable mask;

    public RippleDrawableCompat(Drawable content, Drawable mask) {
        super(content != null ? new Drawable[] { content } : new Drawable[] { } );
        this.mask = mask;
    }

}
```

RippleDrawableCompat

```
public class RippleDrawableCompat extends LayerDrawable {

    private ColorStateList colors;

    public void setColor(ColorStateList colors) {
        this.colors = colors;
    }

}
```

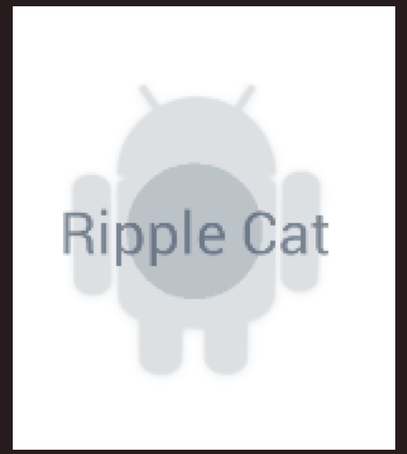
RippleDrawableCompat

```
public class RippleDrawableCompat extends LayerDrawable {

    private float hotspotX;
    private float hotspotY;

    public void setHotspot(float x, float y) {
        this.hotspotX = x;
        this.hotspotY = y;
    }

}
```

RippleDrawableCompat

```
public class RippledImageView extends ImageView {

    ...

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        updateHotspot(event.getX(), event.getY());
        return super.onTouchEvent(event);
    }

    private void updateHotspot(float x, float y) {

        Drawable background = getBackground();
        if (background != null && background instanceof RippleDrawableCompat) {
            ((RippleDrawableCompat) background).setHotspot(x, y);
        }

        Drawable src = getDrawable();
        if (src != null && src instanceof RippleDrawableCompat) {
            ((RippleDrawableCompat) src).setHotspot(x, y);
        }

    }

}
```

Mask



RippleDrawableCompat

```
private Bitmap convertDrawableToBitmap(Drawable mask, Bounds bounds) {

    Bitmap maskBitmap = Bitmap.createBitmap(bounds.width(), bounds.height(), Bitmap.Config.ALPHA_8);

}
```

RippleDrawableCompat

```
private Bitmap convertDrawableToBitmap(Drawable mask, Bounds bounds) {

    Bitmap maskBitmap = Bitmap.createBitmap(bounds.width(), bounds.height(), Bitmap.Config.ALPHA_8);

    Canvas maskCanvas = new Canvas(maskBitmap);
}
```

RippleDrawableCompat

```
private Bitmap convertDrawableToBitmap(Drawable mask, Bounds bounds) {

    Bitmap maskBitmap = Bitmap.createBitmap(bounds.width(), bounds.height(), Bitmap.Config.ALPHA_8);

    Canvas maskCanvas = new Canvas(maskBitmap);

    mask.setBounds(bounds);
    mask.draw(maskCanvas);

    return maskBitmap;
}
```

RippleDrawableCompat

```
public class RippleDrawableCompat extends LayerDrawable  {

    @Override
    protected void onBoundsChange(Rect bounds) {
        super.onBoundsChange(bounds);

        // we have a Drawable to use as a mask
        if (mask != null) {

            Bitmap maskBitmap = convertDrawableToBitmap(mask, bounds)

            // this shader will limit where drawing takes place to only the mask area
            maskShader = new BitmapShader(maskBitmap, Shader.TileMode.CLAMP, Shader.TileMode.CLAMP);

        }

    }
}
```

BitmapShader

RippleDrawableCompat

```
public class RippleDrawableCompat extends LayerDrawable {

    @Override
    protected boolean onStateChange(int[] state) {

        // get the color for the current state
        int color = colorStateList.getColorForState(state, Color.TRANSPARENT);

    }

}
```

RippleDrawableCompat

```
public class RippleDrawableCompat extends LayerDrawable {

    @Override
    protected boolean onStateChange(int[] state) {

        // get the color for the current state
        int color = colorStateList.getColorForState(state, Color.TRANSPARENT);

        LinearGradient gradient = new LinearGradient(0, 0, 0, 0, color, color, Shader.TileMode.CLAMP);
        ComposedShader shader = new ComposeShader(maskShader, gradient, PorterDuff.Mode.SRC_IN);

        paint.setShader(shader);

    }

}
```

Thank You!

getBounds()

The story of Drawables and their View masters

Jamie Huson + Lisa Neigut | 20 Sept 2014