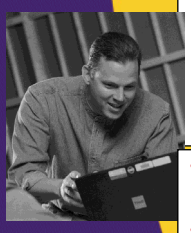
Android Programming: 2D Drawing Part 1: Using onDraw

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/android-tutorial/

© 2012 Marty Hall

For live Android training, please see courses at http://courses.coreservlets.com/.



Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this Android tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 6 or 7 programming, custom mix of topics
  - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, etc.) or survey several
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, EJB3, GWT, Hadoop, SOAP-based and RESTful Web Services

Contact hall@coreservlets.com for details

Topics in This Section

- Extending a View
- Using onDraw
- Referring to custom View in layout file
- Drawing basic shapes with Canvas
- Drawing bitmaps (images) with Canvas

Extending a View

Summary: Java

- **Idea**
  - Extend the View class, put the drawing code in onDraw
    - Note: this assumes the View is placed in a fixed-size region. In lecture on Custom Components, we will discuss using onMeasure to calculate desired sizes.
- **Syntax**

```
public class RandomShapeView extends View {
    …
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        canvas.draw*Blah*(…);
        …
    }
}
```


Summary: XML

- **Idea**
  - Refer to the custom View with the "view" tag in layout file. Pass standard attributes (id, sizes, background).
    - In lecture on Custom Components, we will discuss passing custom attributes.
- **Syntax**

```
…
 <view
      class="com.someCompany.somePackage.YourView"
      android:id="…"
      android:layout_width="…"
      android:layout_height="…"
      android:background="…" />
…
```

If you use regular (non-inner) classes, you can replace <view class="package.Class" …/> with <package.Class …/>. However, this will not work with inner classes.

View Constructors

- **ViewName(Context)**
  - Used when you directly call View constructor. Usually when you add View to Activity with setContentView.

```
public ViewName(Context context) {
    super(context);
    …
}
```

- **ViewName(Context, AttributeSet)**
  - Used when View built from layout XML file. Section on Custom Components will show how to use AttributeSet.

```
public ViewName(Context context, AttributeSet attrs) {
    super(context);
    …
}
```


Using onDraw: Canvas and Paint

- **Canvas**
  - onDraw is passed a Canvas. Use its drawing methods.
- **Paint**
  - Most of the drawing methods (drawArc, drawCircle, etc.) take a Paint as an argument. This is different from Swing and the AWT, where you set the paint on the Graphics separately from the drawing methods.
  - The Paint also incorporates the font size
  - Making a Paint
    - Paint p = new Paint();
    - // Or Paint p = new Paint(Paint.ANTI_ALIAS_FLAG);
    - p.setColor(…);
    - p.setTextSize(…);

Common Canvas Drawing Methods

- **drawColor, drawRGB, drawARGB**
  - Fill entire region with a color. Note that this uses Color, not Paint.
- **drawBitmap**
  - Draw a picture. Create a Bitmap from an image file via BitmapFactory.decodeResource (getResources(),R.drawable.blah)
- **drawCircle, drawOval, drawRect, drawRoundRect, drawArc, drawLine**
  - Draw basic shape, usually inside rectangular region
- **drawText, drawTextOnPath, drawPosText**
  - Draw text in simple or fancy ways. Note that drawPosText is deprecated in recent Android releases.
- **drawPath**
  - A Path lets you combine multiple shapes into one object

Drawing Images

- **Put image files in res/drawable**
  - If file is res/drawable/foo_bar.gif, you will refer to it with R.drawable.foo_bar.
  - You often want versions of your image in various resolutions, so you will put the images in res/drawable-xhdpi (extra high), res/drawable-hdpi (high), res/drawable-mdpi (medium), res/drawable-ldpi (low)
  - Supported image types are jpeg, gif, png, bmp, and (Android 4.0+ only) webp
- **Load image into Bitmap**
  - Bitmap pic = BitmapFactory.decodeResource (getResources(), R.drawable.base_name);
- **Draw Bitmap**
  - canvas.drawBitmap(pic, left, top, null);
    - The last arg is the Paint (null for opaque images)

Triggering Redrawing

- **Automatically**
  - After screen is rotated or covered and reexposed
- **Manually**
  - Yes
    - Call “invalidate” on the View
    - If called by Thread other than main thread, you should call “postInvalidate” instead of “invalidate”
  - No
    - Call “onDraw” directly
  - Very simple animation
    - By repeatedly doing short Thread.sleep, then calling invalidate
      - For details on fancier animation, see http://developer.android.com/guide/topics/graphics/index.html

Avoid Allocating Drawing Objects in onDraw

- **Idea**
  - onDraw is expensive
  - onDraw can be called many times
- **Consequence**
  - Don’t allocate graphics objects in onDraw
  - This is especially true of Bitmaps, but applies to Paint and other simpler objects as well

Example: View that Draws Random Shapes

Example Summary

- **Idea**
  - Make View that draws random backgrounds, circles, rectangles, bitmaps, and text
    - All sizes and locations based on current dimensions of the View
    - View must go in fixed-sized region for now. We will discuss using onMeasure to request sizes in section on Custom Components
  - Redrawing triggered by button in main Activity
    - Gets id of custom View, calls findViewById, then calls invalidate on each button press

View: General Class Structure

```
public class RandomShapeView extends View {
    private Integer[] mBackgrounds = { ... };
    private Paint[] mForegrounds = { ... };
    private Bitmap[] mPics = { ... };
    private String mMessage = "Android";

    public RandomShapeView(Context context) {
        super(context);
    }

    public RandomShapeView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        ...
    }
    ...
}
```

To avoid repeated allocation, objects like Paint and Bitmap objects should be instantiated once per View instantiation (i.e., as instance variable values or in the constructor).

View: onDraw

```
public class RandomShapeView extends View {
    ...
    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        canvas.drawColor(RandomUtils.randomElement(mBackgrounds));
        int viewWidth = getWidth();
        int viewHeight = getHeight();
        int avgShapeWidth = viewWidth/5;
        for(int i=0; i<20; i++) {
            drawRandomCircle(canvas, viewWidth,
                             viewHeight, avgShapeWidth);
            drawRandomRect(canvas, viewWidth,
                           viewHeight, avgShapeWidth);
            drawRandomBitmap(canvas, viewWidth, viewHeight);
            drawRandomText(canvas, viewWidth,
                           viewHeight, avgShapeWidth);
        }
    }
}
```

You must call getWidth and getHeight in onDraw (or after), not in the constructor. However, to avoid repeated allocation, "real" objects like Paint and Bitmap objects should not be instantiated in onDraw.

View: Random Background Colors

```
public class RandomShapeView extends View {
    private Integer[] mBackgrounds =
        { Color.CYAN, Color.GRAY, Color.LTGRAY,
          Color.MAGENTA, Color.YELLOW, Color.WHITE };
    ...

   @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        canvas.drawColor(RandomUtils.randomElement(mBackgrounds));
        ...
    }
}
```

Notice that in Android, colors are really ints, as opposed to AWT and Swing where colors are real Objects. Also note that you draw backgrounds with colors, not Paints.

View: Random Circles

```
public class RandomShapeView extends View {
    private Paint[] mForegrounds =
        { makePaint(Color.BLACK), makePaint(Color.BLUE),
          makePaint(Color.GREEN), makePaint(Color.RED) };
   ...

    private void drawRandomCircle(Canvas canvas, int viewWidth,
                                  int viewHeight, int avgShapeWidth) {
        float x = RandomUtils.randomFloat(viewWidth);
        float y = RandomUtils.randomFloat(viewHeight);
        float radius = RandomUtils.randomFloat(avgShapeWidth/2);
        Paint circleColor = RandomUtils.randomElement(mForegrounds);
        canvas.drawCircle(x, y, radius, circleColor);
    }

    private Paint makePaint(int color) {
        Paint p = new Paint();
        p.setColor(color);
        return(p);
    }
}
```

drawRandomCircle is called from the loop in onDraw.

Note that the Paint objects are not allocated in onDraw, but rather just once per class instantiation.

View: Random Rectangles

```
public class RandomShapeView extends View {
    private Paint[] mForegrounds =
        { makePaint(Color.BLACK), makePaint(Color.BLUE),
          makePaint(Color.GREEN), makePaint(Color.RED) };

   ...

    private void drawRandomRect(Canvas canvas, int viewWidth,
                                int viewHeight, int avgShapeWidth) {
        float left = RandomUtils.randomFloat(viewWidth);
        float top = RandomUtils.randomFloat(viewHeight);
        float width = RandomUtils.randomFloat(avgShapeWidth);
        float right = left + width;
        float bottom = top + width;
        Paint squareColor = RandomUtils.randomElement(mForegrounds);
        canvas.drawRect(left, top, right, bottom, squareColor);
    }
}
```


View: Random Bitmaps

```
public class RandomShapeView extends View {
   private Bitmap[] mPics =
        { makeBitmap(R.drawable.emo_im_angel),
          makeBitmap(R.drawable.emo_im_cool),
          makeBitmap(R.drawable.emo_im_crying),
          makeBitmap(R.drawable.emo_im_happy),
          makeBitmap(R.drawable.emo_im_yelling) };
   ...

   private void drawRandomBitmap(Canvas canvas, int viewWidth,
                                 int viewHeight) {
        float left = RandomUtils.randomFloat(viewWidth);
        float top = RandomUtils.randomFloat(viewHeight);
        Bitmap pic = RandomUtils.randomElement(mPics);
        // Last arg is the Paint: you can use null for opaque images
        canvas.drawBitmap(pic, left, top, null);
    }

    private Bitmap makeBitmap(int bitmapId) {
        return(BitmapFactory.decodeResource(getResources(), bitmapId));
    }
}
```

There are versions of em_im_angel.png, emo_im_cool.png, etc., in res/drawable/xhdpi (extra-high dpi), res/drawable/hdpi (high dpi), res/drawable/mdpi (medium dpi), and res/drawable/ldpi (low dpi).

Note again that the bitmaps are allocated once per View instantiation, not repeatedly in onDraw.

Random Text

```
public class RandomShapeView extends View {
   private String mMessage = "Android";

   ...

    private void drawRandomText(Canvas canvas, int viewWidth,
                                int viewHeight, int avgShapeWidth) {
        float x = RandomUtils.randomFloat(viewWidth);
        float y = RandomUtils.randomFloat(viewHeight);
        float textSize = RandomUtils.randomFloat(avgShapeWidth);
        Paint textPaint = RandomUtils.randomElement(mForegrounds);
        textPaint.setTextSize(textSize);
        canvas.drawText(mMessage, x, y, textPaint);
    }
}
```


Helper Class: Random Utilities

```
public class RandomUtils {
    private static Random r = new Random();

    public static int randomInt(int range) {
        return(r.nextInt(range));
    }

    public static int randomIndex(Object[] array) {
        return(randomInt(array.length));
    }

    public static <T> T randomElement(T[] array) {
        return(array[randomIndex(array)]);
    }

    public static float randomFloat(int n) {
        return((float)Math.random()*n);
    }
}
```

Layout File for Example (activity_draw_shapes1.xml)

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:onClick="redraw"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/redraw_button_label" />
    <view
        class="com.coreservlets.drawing.RandomShapeView"
        android:id="@+id/drawing_area"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```


Activity for Example

```
public class DrawShapes1 extends Activity {
    private RandomShapeView mDrawingArea;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_draw_shapes1);
        mDrawingArea  =
            (RandomShapeView)findViewById(R.id.drawing_area);
    }

    /** Handles events for the button. Redraws the ShapeView. */

    public void redraw(View clickedButton) {
        mDrawingArea.invalidate();
    }
}
```

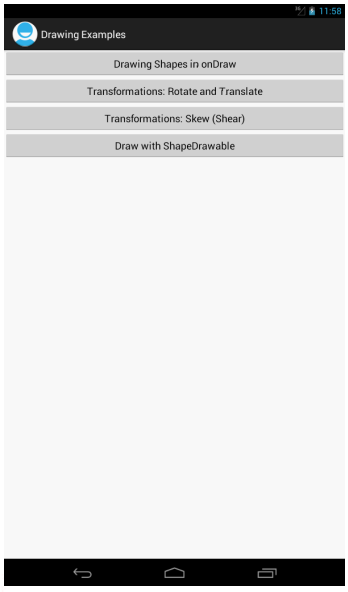
Overall Main Layout File (main.xml)

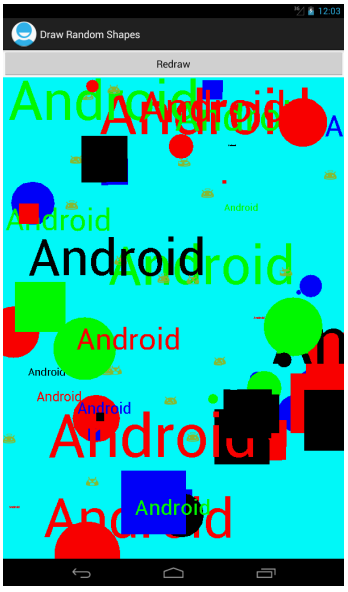
```
<LinearLayout ... android:orientation="vertical">
    <Button
        android:onClick="launchDrawShapes1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/random_shape_button_label" />
    <Button
        android:onClick="launchRotate"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/rotate_button_label" />
    <Button
        android:onClick="launchSkew"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/skew_button_label" />
    <Button
        android:onClick="launchDrawShapes2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/shape_drawable_button_label" />
</LinearLayout>
```


Overall Main Activity

```
public class DrawingExampleLauncher extends Activity {

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

    public void launchDrawShapes1(View clickedButton) {
        Intent activityIntent =
                new Intent(this, DrawShapes1.class);
        startActivity(activityIntent);
    }

   // Similar button-handling code for
   // other three drawing examples

}
```

Results





Wrap-Up

References

- **Android Dev Guide: "Canvas & Drawables"**
  - http://developer.android.com/guide/topics/graphics/2d-graphics.html
- ***Pro Android 3* (Komatineni et al)**
  - Chapter 16, "Exploring 2D Animation"
- ***Professional Android 4* (Meier)**
  - "Creating New Views" in Chapter 4
  - "Enhancing Your Views" in Chapter 11
- ***Programming Android* (Mednieks et al)**
  - Chapter 9, "Drawing 2D and 3D Graphics"

Summary

- **Java**
  - Extend View
  - Make 2 constructors (Context and Context/AttributeSet)
  - Override onDraw

```
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    canvas.drawBlah(…);
    …
}
```

  - Avoid allocating Paint, Bitmap, etc. in onDraw
- **XML**

```
<view
    class="com.someCompany.somePackage.YourView"
    android:id="…"
    android:layout_width="…"
    android:layout_height="…"
    android:background="…" />
```

Questions?