Wrocław University of Technology

ANDROID (4)

2D Graphics and Animation, Handling Screen Rotation

Marek Piasecki

Outline

- 2D graphics drawing
  - Color / Paint / Canvas
  - XML drawable (from resources)
  - direct to a Canvas / View.onDraw()
- 2D animation
  - frame by frame
  - XML / View animation
- Handling screen rotation
  - blocking rotation
  - automatic handling (separate vertical/landscape layouts)
  - preserving temporary data

android.graphics.Color (1)

- colors are represented with four numbers (ARGB):
  - alpha,
  - red,
  - green,
  - blue
- Each component can have 256 possible values, or 8 bits, typically packed into a 32-bit integer (for efficiency)
- Alpha is a measure of transparency:
  - value "0" -> color is completely transparent
  - value "255" -> color is completely opaque
  - in the middle -> semitransparent colors
- color definition:
  - use one of the static constants on the Color class
    int color = Color.BLUE; // solid blue

android.graphics.Color (2)

- static factory methods of the Color

```
int color1 = Color.argb(127, 255, 0, 255);  // Translucent purple
int color2 = Color.rgb(255, 0, 0);          //opaque (alpha=255) red
```

- better off defining all your colors in an XML resource

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <color name="mycolor">#7fff00ff</color>
</resources>
```

- then can reference colors
  by defined name in other XML files
- or in Java code

```
int color = getResources().getColor(R.color.mycolor);
```

android.graphics.Paint / Canvas

- Paint class:
  - holds the style and color
  - to draw a solid color:
    paint.setColor(Color.LTGRAY);

- Canvas class
  - represents a surface on which you draw
  - Initially clear, like blank transparencies for an overhead projector
  - Canvas methods let draw lines, rectangles, circles, or other arbitrary graphics

In Android:

Display screen → Activity → View → View.onDraw(Canvas) → **Canvas**

XML Drawables (1)

Draw into a View object from your layout:

- drawing is handled by the system (normal View hierarchy drawing process)
- simply define the graphics in XML

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    type="oval" >

    <solid android:color="#00000000" />

    <padding
        android:bottom="4sp"
        android:left="10sp"
        android:right="10sp"
        android:top="4sp" />

    <stroke
        android:width="1dp"
        android:color="#FFFFFFFF" />

</shape>
```

XML Drawables (View's Layout definition) (2)

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" >

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >

        <ImageView
            android:layout_width="fill_parent"
            android:layout_height="50dip"
            android:src="@drawable/shape_1" />

        <ImageView
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:src="@drawable/line" />
```

XML Drawables (3)

DEMO

from „Android in Action”
example source

chapter 9 / XMLDraw

Direct drawing on Canvas (1)

```
public class Graphics extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(new GraphicsView(this));
    }

    static public class GraphicsView extends View {
        public GraphicsView(Context context) {
            super(context);
        }
        @Override
        protected void onDraw(Canvas canvas) {
            // Drawing commands go here
        }
    }
```

android.graphics.Canvas (3)

Canvas methods drawing geometric primitives:

public void **drawColor** (int color)

public void **drawPaint** (Paint paint)

public void **drawArc** (RectF oval, float startAngle, float sweepAngle, boolean useCenter, Paint paint)

public void **drawCircle** (float cx, float cy, float radius, Paint paint)

public void **drawLine** (float startX, float startY, float stopX, float stopY, Paint paint)

public void **drawRect** (Rect r, Paint paint)

public void **drawText** (String text, float x, float y, Paint paint)

Canvas drawing example (1)

```
protected void onDraw(Canvas c){
    super.onDraw(c);

    Paint paint = new Paint();
    paint.setStyle(Paint.Style.FILL);
    // make the entire canvas white
    paint.setColor(Color.WHITE);
    c.drawPaint(paint);

    paint.setAntiAlias(true);
    paint.setColor(Color.BLUE);
    c.drawCircle(80, 20, 15, paint);

    paint.setColor(Color.GREEN);
    c.drawRect(20, 5, 50, 100, paint);


    paint.setColor(Color.RED);
    paint.setStyle(Paint.Style.FILL);
    paint.setAntiAlias(true);
    paint.setTextSize(30);
    c.drawText("Me like painting!", 30, 200, paint);
```

Canvas drawing example (2)

```
int x = 75; // x increases from left to right
int y = 140; // y increases from top to bottom
paint.setColor(Color.GRAY);
paint.setTextSize(25);
String str2rotate = "Rotated text!";
// draw bounding rect before rotating text
Rect rect = new Rect();
// rotate the canvas on center of the text to draw
c.rotate(-45, x + rect.exactCenterX(), y + rect.exactCenterY());
// draw the rotated text
paint.setStyle(Paint.Style.FILL);
c.drawText(str2rotate, x, y, paint);
//undo the rotate
c.restore();

Resources res = this.getResources();
Bitmap bitmap = BitmapFactory.decodeResource(res, R.drawable.icon);
c.drawBitmap(bitmap,90 ,200 , paint);
```

Frame by frame animation (1)

- simple animations by showing a set of images one after another
- define a set of resources in a XML file and call AnimationDrawable start().
- create a new directory called /anim under the /res resources directory. Place all the images for this example in the /drawable directory.
- DEMO: [XMLAnimate] (chapter 9)

Frame by frame animation (2)

- create a new directory called **/anim**
  under the **/res** resources directory.
  Place all the images for this example in the **/drawable** directory.
- create an XML file **Simple_animation.xml**

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
 id="selected" android:oneshot="false">
        <item android:drawable="@drawable/ball1" android:duration="50" />
        <item android:drawable="@drawable/ball2" android:duration="50" />
        <item android:drawable="@drawable/ball3" android:duration="50" />
        <item android:drawable="@drawable/ball4" android:duration="50" />
        <item android:drawable="@drawable/ball5" android:duration="50" />
        <item android:drawable="@drawable/ball6" android:duration="50" />
   </animation-list>
```

- two attributes:
  - **drawable**, which describes the path to the image,
  - **duration** - the length of time to show the image, in miliseconds

Frame by frame animation (3)

- we can't control the animation from within the OnCreate method !

```
public class XMLAnimate extends Activity {
    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.main);
        ImageView img = (ImageView) findViewById(R.id.simple_anim);
        img.setBackgroundResource(R.anim.simple_animation);

        MyAnimationRoutine mar = new MyAnimationRoutine();
        Timer t = new Timer(false);
        t.schedule(mar, 100);
    }

    class MyAnimationRoutine extends TimerTask {
        @Override
        public void run() {
            ImageView img = (ImageView) findViewById(R.id.simple_anim);
            AnimationDrawable frameAnimation =
                                (AnimationDrawable) img.getBackground();
            frameAnimation.start();
        }
    }
```

Frame by frame animation (4)

DEMO

from „Android in Action”
example source

chapter 9 / XMLAnimate

XML View animation (1)

- define animations that can rotate, fade, move, or stretch graphics or text
- Animation XML files are placed in the res/anim source directory.

- Android supports four types of animations:
  - **<alpha>**—Defines fading, from 0.0 to 1.0 (0.0 being transparent)
  - **<scale>**—Defines sizing, x and y (1.0 being no change)
  - **<translate>**—Defines motion, x and y (percentage or absolute)
  - **<rotate>**—Defines rotation, pivot from x and y (degrees)

- attributes that can be used with any animation type:
  - **duration** — Duration, in milliseconds
  - **startOffset** — Offset start time, in milliseconds
  - **interpolator** — Used to define a velocity curve for speed of animation

XML View animation (2)

- example parameters for "scale" (scaler.xml resource)

```
android:fromXScale = "0.5"
android:toXScale = "2.0"
android:fromYScale = "0.5"
android:toYScale = "2.0"
android:pivotX = "50%"
android:pivotY = "50%"
android:startOffset = "700"
android:duration = "400"
android:fillBefore = "false"
```

- can use this animation with any View object:

```
view.startAnimation(AnimationUtils.loadAnimation(this, R.anim.scaler));
```

Getting screen sizes

A common mistake

→ to use the **width** and **height** of a view inside its constructor!

- When a view's constructor is called, Android doesn't know yet how big the view will be, so the sizes are set to zero.
- The real sizes are calculated during the layout stage (which occurs after construction but before anything is drawn)
- Use the onSizeChanged( )method to be notified of the values when they are known
- Use the getWidth( ) and getHeight( )methods in the onDraw( ) method

```
@Override
protected void onSizeChanged(int w, int h, int oldw, int oldh) {
      width = w / 9f;
      height = h / 9f;
      getRect(selX, selY, selRect);
      Log.d(TAG, "onSizeChanged: width " + width + ", height " + height);
      super.onSizeChanged(w, h, oldw, oldh);
}
```

Handling screen rotation (1)

- To block Android from rotating your activity, add
  android:screenOrientation = "portrait" (or "landscape")
  to AndroidManifest.xml file:

```
<activity android:name=".RotationDemo"
      android:screenOrientation="portrait"
      android:label="@string/app_name">
      <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
</activity>
```

- add android:screenOrientation = "sensor"
  to rotate screen based on the position of the phone from accelerometer
  (this setting disables having the keyboard trigger a rotation event)

Handling screen rotation (2)

Customize/provide different layouts for various display orientations by creating specific folders for each configuration:

- **res/layout** - default layout folder,
- **res/layout-port** - support the portrait orientation,
- **res/layout-land** - for landscape orientation,
- **res/layoutsquare** - for square displays

Example layout definition downloaded for horizontal orientation

res/layout-land/main.xml

Handling screen rotation (3)

- can set the orientation of the device in code:
  setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
- To handle rotations on your own:
  - put an android:configChanges entry in your AndroidManifest.xml listing the configuration changes you want to handle yourself
    android:configChanges="orientation|keyboardHidden„
  - implement onConfigurationChanged()
    which will be called when configuration changes you listed in occurs:

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);

    if (newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE) {
        Toast.makeText(this, "landscape", Toast.LENGTH_SHORT).show();
    } else if (newConfig.orientation == Configuration.ORIENTATION_PORTRAIT){
        Toast.makeText(this, "portrait", Toast.LENGTH_SHORT).show();
    }
}
```

Handling screen rotation (4)

Preserving temporary data durring configuration changes:

(by default Android destroys and re-creates your activity from scratch)

1. blocking configuration changes (android:screenOrientation=…)
2. handle rotations on your own (android:configChanges= …)
   by implementing onConfigurationChanged(Configuration);
3. preserve/store the data in the **Boundle** by implementing
   onSaveInstanceState(Boundle);
   onCreate(Boundle); or onRestoreInstanceState(Boundle);
   data have to be serialisable!
4. preserve/store data in any **Object** by implementing
   void onRetainNonConfigurationInstance(Object)
   Object getLastNonConfigurationInstance( );
   data could be any Java object.