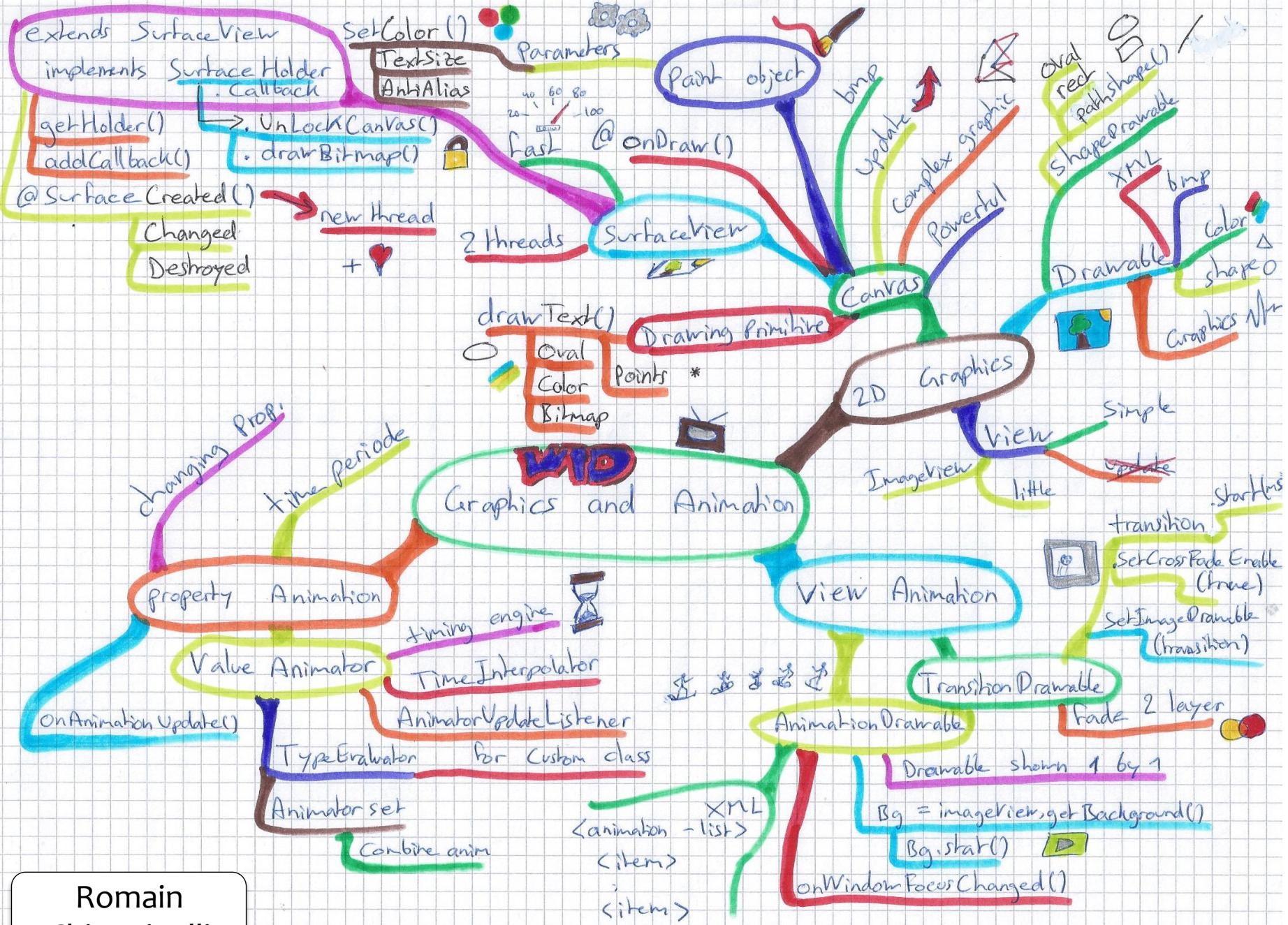


Romain
Chiappinelli



2D GRAPHICS
&
ANIMATION

TOPICS

2D GRAPHICS

IMAGEVIEW

CANVAS

VIEW ANIMATION

PROPERTY ANIMATION

DRAWING 2D GRAPHICS

DRAW TO A VIEW

SIMPLE GRAPHICS, LITTLE OR NO UPDATING

DRAW TO A CANVAS

MORE COMPLEX GRAPHICS, WITH REGULAR
UPDATES

DRAWABLE

SOMETHING THAT CAN BE DRAWN, SUCH AS A
BITMAP, COLOR, SHAPE, ETC.

EXAMPLES:

BITMAPDRAWABLE

SHAPEDRAWABLE

COLORDRAWABLE

DRAWING TO VIEWS

CAN SET DRAWABLE OBJECTS ON VIEWS

CAN DO THIS VIA XML OR
PROGRAMMATICALLY

GRAPHICS BUBBLE

APPLICATIONS DISPLAY A SINGLE IMAGEVIEW

IMAGEVIEW HOLDS AN IMAGE OF A BUBBLE



GRAPHICS BUBBLE XML

```
<ImageView  
    android:id="@+id/imageView1"  
    android:layout_width="250dp"  
    android:layout_height="250dp"  
    android:layout_centerInParent="true"  
    android:contentDescription="@string/bubble_desc"  
    android:src="@drawable/b128" />
```

GRAPHICS BUBBLE PROGRAM

```
public class BubbleActivity extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
  
        RelativeLayout relativeLayout = (RelativeLayout) findViewById(R.id.frame);  
  
        ImageView bubbleView = new ImageView(getApplicationContext());  
        bubbleView  
            .setImageDrawable(getResources().getDrawable(R.drawable.b128));  
  
        int width = (int) getResources().getDimension(R.dimen.image_width);  
        int height = (int) getResources().getDimension(R.dimen.image_height);  
  
        RelativeLayout.LayoutParams params = new RelativeLayout.LayoutParams(  
            width, height);  
        params.addRule(RelativeLayout.CENTER_IN_PARENT);  
  
        bubbleView.setLayoutParams(params);  
  
        relativeLayout.addView(bubbleView);  
    }  
}
```

SHAPEDRAWABLE

USED FOR DRAWING PRIMITIVE SHAPES

SHAPE REPRESENTED BY A SHAPE CLASS

PATHSHAPE - LINES

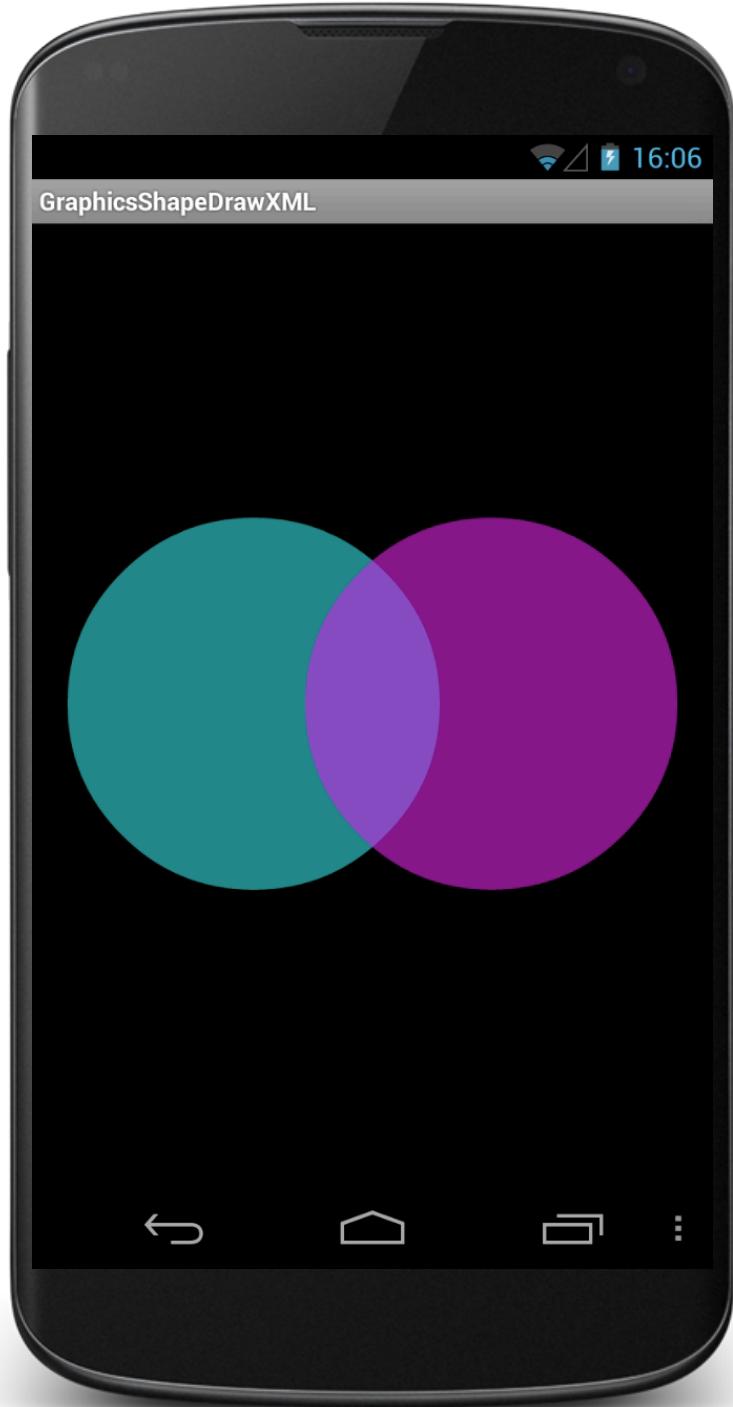
RECTSHAPE - RECTANGLES

OVALSHAPE - OVALS & RINGS

GRAPHICS SHAPEDRAW

APPLICATIONS DISPLAY TWO SHAPES WITHIN A
RELATIVELAYOUT

THE TWO SHAPES ARE PARTIALLY
OVERLAPPING AND SEMI-TRANSPARENT



GRAPHICS SHAPEDRAW

```
public class ShapeDrawActivity extends Activity {
    int alpha = 127;

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

        int width = (int) getResources().getDimension(R.dimen.image_width);
        int height = (int) getResources().getDimension(R.dimen.image_height);
        int padding = (int) getResources().getDimension(R.dimen.padding);

        // Get container View
        RelativeLayout rl = (RelativeLayout) findViewById(R.id.main_window);
```

GRAPHICS SHAPEDRAW

```
// Create Cyan Shape
ShapeDrawable cyanShape = new ShapeDrawable(new OvalShape());
cyanShape.getPaint().setColor(Color.CYAN);
cyanShape.setIntrinsicHeight(height);
cyanShape.setIntrinsicWidth(width);
cyanShape.setAlpha(alpha);

// Put Cyan Shape into an ImageView
ImageView cyanView = new ImageView(getApplicationContext());
cyanView.setImageDrawable(cyanShape);
cyanView.setPadding(padding, padding, padding, padding);

// Specify placement of ImageView within RelativeLayout
RelativeLayout.LayoutParams cyanViewLayoutParams = new RelativeLayout.LayoutParams(
    height, width);
cyanViewLayoutParams.addRule(RelativeLayout.CENTER_VERTICAL);
cyanViewLayoutParams.addRule(RelativeLayout.ALIGN_PARENT_LEFT);
cyanView.setLayoutParams(cyanViewLayoutParams);
rl.addView(cyanView);
```

GRAPHICS SHAPEDRAW

```
// Create Magenta Shape
ShapeDrawable magentaShape = new ShapeDrawable(new OvalShape());
magentaShape.getPaint().setColor(Color.MAGENTA);
magentaShape.setIntrinsicHeight(height);
magentaShape.setIntrinsicWidth(width);
magentaShape.setAlpha(alpha);

// Put Magenta Shape into an ImageView
ImageView magentaView = new ImageView(getApplicationContext());
magentaView.setImageDrawable(magentaShape);
magentaView.setPadding(padding, padding, padding, padding);

// Specify placement of ImageView within RelativeLayout
RelativeLayout.LayoutParams magentaViewLayoutParams = new RelativeLayout.LayoutParams(
    height, width);
magentaViewLayoutParams.addRule(RelativeLayout.CENTER_VERTICAL);
magentaViewLayoutParams.addRule(RelativeLayout.ALIGN_PARENT_RIGHT);

magentaView.setLayoutParams(magentaViewLayoutParams);

rl.addView(magentaView);
```

DRAWING WITH A CANVAS

A BITMAP (A MATRIX OF PIXELS)

A CANVAS FOR DRAWING TO THE UNDERLYING
BITMAP

A DRAWING PRIMITIVE (E.G. RECT,
PATH, TEXT, BITMAP)

A PAINT OBJECT (FOR SETTING DRAWING
COLORS & STYLES)

DRAWING PRIMITIVES

CANVAS SUPPORTS MULTIPLE DRAWING METHODS

DRAWTEXT()

DRAWPOINTS()

DRAWCOLOR()

DRAWOVAL()

DRAWBITMAP()

PAINT

SPECIFIES STYLE PARAMETERS FOR DRAWING,
E.G.,

SETSTROKEWIDTH()

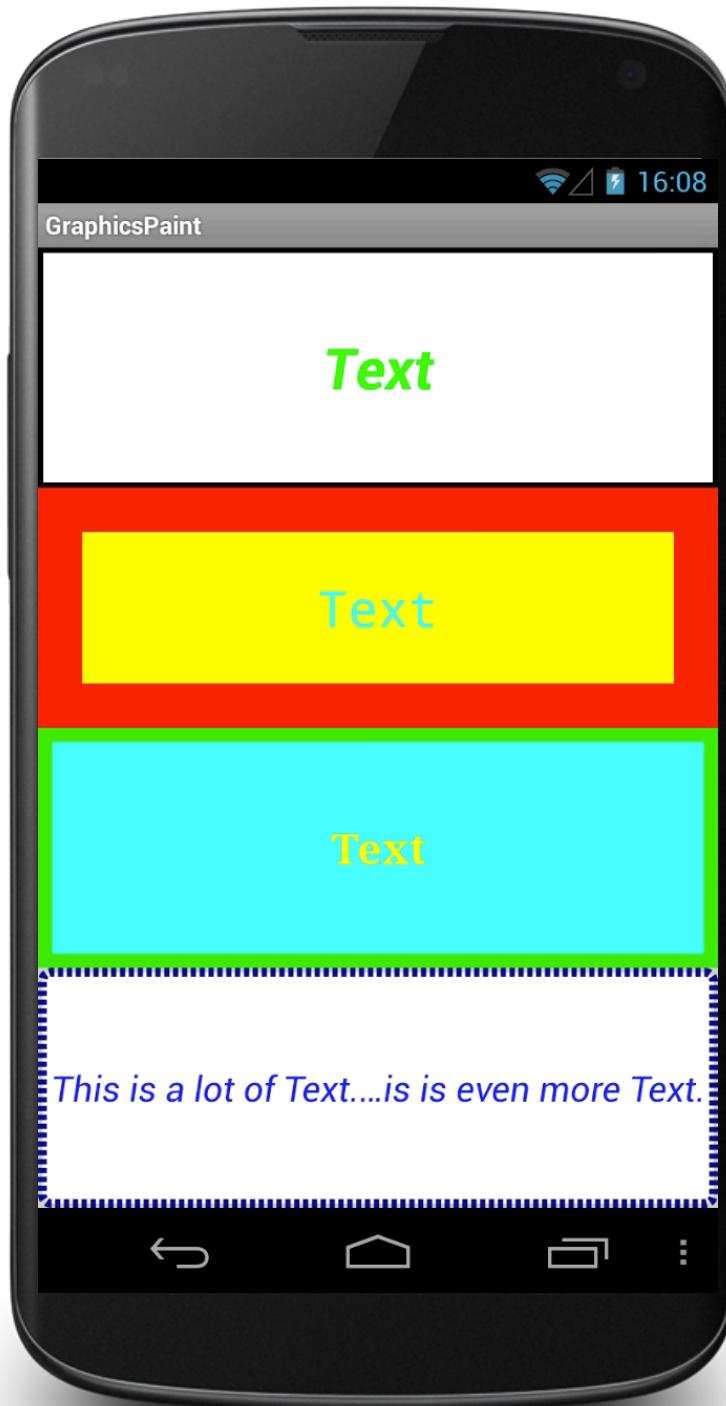
SETTEXTSIZE()

SETCOLOR()

SETANTI_ALIAS()

GRAPHICS PAINT

APPLICATION DRAWS SEVERAL BOXES HOLDING
TEXT, SO USING DIFFERENT PAINT SETTINGS
EACH TIME



GRAPHICS PAINT

```
<TextView  
    android:id="@+id/textView1"  
    android:layout_width="match_parent"  
    android:layout_height="0dp"  
    android:layout_weight="1"  
    android:background="@drawable/sq1"  
    android:gravity="center"  
    android:text="@string/text_literal"  
    android:textColor="#ff00ff00"  
    android:textSize="32sp"  
    android:textStyle="bold/italic"  
    android:typeface="normal" />
```

GRAPHICS PAINT

```
<TextView  
    android:id="@+id/imageView2"  
    android:layout_width="match_parent"  
    android:layout_height="0dp"  
    android:layout_weight="1"  
    android:background="@drawable/sq2"  
    android:gravity="center"  
    android:text="@string/text_literal"  
    android:textColor="#FF00FFFF"  
    android:textSize="28sp"  
    android:textStyle="normal"  
    android:typeface="monospace" />
```

GRAPHICS PAINT

```
<TextView  
    android:id="@+id/imageView3"  
    android:layout_width="match_parent"  
    android:layout_height="0dp"  
    android:layout_weight="1"  
    android:background="@drawable/sq3"  
    android:gravity="center"  
    android:text="@string/text_literal"  
    android:textColor="#FFFFFF00"  
    android:textSize="24sp"  
    android:textStyle="bold"  
    android:typeface="serif" />
```

GRAPHICS PAINT

```
<TextView  
    android:id="@+id/imageView4"  
    android:layout_width="match_parent"  
    android:layout_height="0dp"  
    android:layout_weight="1"  
    android:background="@drawable/sq4"  
    android:ellipsize="middle"  
    android:gravity="center"  
    android:singleLine="true"  
    android:text="@string/long_text"  
    android:textColor="#FF0000FF"  
    android:textSize="20sp"  
    android:textStyle="italic"  
    android:typeface="sans" />
```

DRAWING WITH A CANVAS

CAN DRAW TO GENERIC VIEWS, OR TO
SURFACEVIEWS

DRAWING TO VIEWS

USE WHEN UPDATES ARE INFREQUENT

CREATE A CUSTOM VIEW CLASS

SYSTEM PROVIDES THE CANVAS TO THE VIEW
WHEN IT CALLS THE VIEW'S ONDRAW()
METHOD

DRAWING TO SURFACEVIEWS

CREATE A CUSTOM SURFACEVIEW

PROVIDE SECONDARY THREAD FOR DRAWING

APPLICATION PROVIDES ITS OWN CANVAS AND
HAS GREATER CONTROL OVER DRAWING

GRAPHICS BUBBLE

THIS APPLICATION DRAWS TO CUSTOM VIEW
IT HAS AN INTERNAL THREAD THAT
PERIODICALLY WAKES UP AND CAUSES THE
VIEW TO MOVE AND TO BE REDRAWN



GRAPHICS CANVAS BUBBLE

```
public class BubbleActivity extends Activity {
    protected static final String TAG = "BubbleActivity";

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

        final RelativeLayout frame = (RelativeLayout) findViewById(R.id.frame);
        final Bitmap bitmap = BitmapFactory.decodeResource(getResources(),
            R.drawable.b128);
        final BubbleView bubbleView = new BubbleView(getApplicationContext(),
            bitmap);

        frame.addView(bubbleView);

        new Thread(new Runnable() {
            @Override
            public void run() {
                while (bubbleView.move()) {
                    bubbleView.postInvalidate();
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        Log.i(TAG, "InterruptedException");
                    }
                }
            }
        }).start();
    }
}
```

GRAPHICS CANVAS BUBBLE

```
private class BubbleView extends View {

    private static final int STEP = 100;
    final private Bitmap mBitmap;

    private Coords mCurrent;
    final private Coords mDxDy;

    final private DisplayMetrics mDisplayMetrics;
    final private int mDisplayWidth;
    final private int mDisplayHeight;
    final private int mBitmapWidthAndHeight, mBitmapWidthAndHeightAdj;
    final private Paint mPainter = new Paint();
```

GRAPHICS CANVAS BUBBLE

```
public BubbleView(Context context, Bitmap bitmap) {  
    super(context);  
  
    mBitmapWidthAndHeight = (int) getResources().getDimension(  
        R.dimen.image_height);  
    this.mBitmap = Bitmap.createScaledBitmap(bitmap,  
        mBitmapWidthAndHeight, mBitmapWidthAndHeight, false);  
  
    mBitmapWidthAndHeightAdj = mBitmapWidthAndHeight + 20;  
  
    mDisplayMetrics = new DisplayMetrics();  
    BubbleActivity.this.getWindowManager().getDefaultDisplay()  
        .getMetrics(mDisplayMetrics);  
    mDisplayWidth = mDisplayMetrics.widthPixels;  
    mDisplayHeight = mDisplayMetrics.heightPixels;
```

GRAPHICS CANVAS BUBBLE

```
Random r = new Random();
float x = (float) r.nextInt(mDisplayWidth);
float y = (float) r.nextInt(mDisplayHeight);
mCurrent = new Coords(x, y);

float dy = (float) r.nextInt(mDisplayHeight) / mDisplayHeight;
dy *= r.nextInt(2) == 1 ? STEP : -1 * STEP;
float dx = (float) r.nextInt(mDisplayWidth) / mDisplayWidth;
dx *= r.nextInt(2) == 1 ? STEP : -1 * STEP;
mDxDy = new Coords(dx, dy);

mPainter.setAntiAlias(true);

}
```

GRAPHICSCANVASBUBBLE

```
@Override
protected void onDraw(Canvas canvas) {
    Coords tmp = mCurrent.getCoords();
    canvas.drawBitmap(mBitmap, tmp.mX, tmp.mY, mPainter);
}

protected boolean move() {
    mCurrent = mCurrent.move(mDxDy);

    if (mCurrent.mY < 0 - mBitmapWidthAndHeightAdj
        || mCurrent.mY > mDisplayHeight + mBitmapWidthAndHeightAdj
        || mCurrent.mX < 0 - mBitmapWidthAndHeightAdj
        || mCurrent.mX > mDisplayWidth + mBitmapWidthAndHeightAdj) {
        return false;
    } else {
        return true;
    }
}
```

CANVAS WITH SURFACEVIEW

USED FOR MORE HIGH-PERFORMANCE
DRAWING OUTSIDE THE UI THREAD

SURFACEVIEW

SURFACEVIEW MANAGES A LOW-LEVEL
DRAWING AREA CALLED A SURFACE

THE SURFACE REPRESENT A DRAWING AREA
WITHIN THE VIEW HIERARCHY

DEFINING A CUSTOM SURFACEVIEW

SUBCLASS SURFACEVIEW & IMPLEMENT
SURFACEHOLDER.CALLBACK

SURFACEHOLDER.CALLBACK DECLARES
LIFECYCLE METHODS THAT ARE CALLED
WHEN THE SURFACE CHANGES

USING A SURFACEVIEW

SET UP SURFACEVIEW

DRAW TO SURFACEVIEW

SETUP

USE SURFACEVIEW'S GETHOLDER() TO
ACQUIRE SURFACE

SETUP

REGISTER FOR CALLBACKS WITH
SURFACEHOLDER'S ADDCALLBACK()

SURFACECREATE()

SURFACECHANGED()

SURFACEDESTROYED()

SETUP

CREATE THE THREAD ON WHICH DRAWING OPERATIONS WILL EXECUTE

DRAWING

ACQUIRE LOCK ON CANVAS

SURFACEHOLDER.LOCKCANVAS()

DRAW

CANVAS.DRAWBITMAP()

UNLOCK CANVAS

SURFACEHOLDER.UNLOCKCANVASANDPOST()



GRAPHICS CANVAS BUBBLE SURFACEVIEW

```
public class BubbleActivity extends Activity {  
  
    BubbleView mBubbleView;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
  
        RelativeLayout relativeLayout = (RelativeLayout) findViewById(R.id.frame);  
        final BubbleView bubbleView = new BubbleView(getApplicationContext(),  
            BitmapFactory.decodeResource(getResources(), R.drawable.b128));  
  
        relativeLayout.addView(bubbleView);  
    }  
}
```

GRAPHICS CANVAS BUBBLE SURFACEVIEW

```
private class BubbleView extends SurfaceView implements  
    SurfaceHolder.Callback {  
  
    private final Bitmap mBitmap;  
    private final int mBitmapHeightAndWidth, mBitmapHeightAndWidthAdj;  
    private final DisplayMetrics mDisplay;  
    private final int mDisplayWidth, mDisplayHeight;  
    private float mX, mY, mDx, mDy, mRotation;  
    private final SurfaceHolder mSurfaceHolder;  
    private final Paint mPainter = new Paint();  
    private Thread mDrawingThread;  
  
    private static final int MOVE_STEP = 1;  
    private static final float ROT_STEP = 1.0f;
```

GRAPHICS CANVAS BUBBLE SURFACEVIEW

```
public BubbleView(Context context, Bitmap bitmap) {
    super(context);

    mBitmapHeightAndWidth = (int) getResources().getDimension(
        R.dimen.image_height_width);
    this.mBitmap = Bitmap.createScaledBitmap(bitmap,
        mBitmapHeightAndWidth, mBitmapHeightAndWidth, false);

    mBitmapHeightAndWidthAdj = mBitmapHeightAndWidth / 2;

    mDisplay = new DisplayMetrics();
    BubbleActivity.this.getWindowManager().getDefaultDisplay()
        .getMetrics(mDisplay);
    mDisplayWidth = mDisplay.widthPixels;
    mDisplayHeight = mDisplay.heightPixels;
```

GRAPHICS CANVAS BUBBLE SURFACEVIEW

```
Random r = new Random();
mX = (float) r.nextInt(mDisplayHeight);
mY = (float) r.nextInt(mDisplayWidth);
mDx = (float) r.nextInt(mDisplayHeight) / mDisplayHeight;
mDx *= r.nextInt(2) == 1 ? MOVE_STEP : -1 * MOVE_STEP;
mDy = (float) r.nextInt(mDisplayWidth) / mDisplayWidth;
mDy *= r.nextInt(2) == 1 ? MOVE_STEP : -1 * MOVE_STEP;
mRotation = 1.0f;

mPainter.setAntiAlias(true);

mSurfaceHolder = getHolder();
mSurfaceHolder.addCallback(this);
}
```

GRAPHICS CANVAS BUBBLE SURFACEVIEW

```
private void drawBubble(Canvas canvas) {
    canvas.drawColor(Color.DKGRAY);
    mRotation += ROT_STEP;
    canvas.rotate(mRotation, mY + mBitmapHeightAndWidthAdj, mX
                  + mBitmapHeightAndWidthAdj);
    canvas.drawBitmap(mBitmap, mY, mX, mPainter);
}

private boolean move() {
    mX += mDx;
    mY += mDy;
    if (mX < 0 - mBitmapHeightAndWidth
        || mX > mDisplayHeight + mBitmapHeightAndWidth
        || mY < 0 - mBitmapHeightAndWidth
        || mY > mDisplayWidth + mBitmapHeightAndWidth) {
        return false;
    } else {
        return true;
    }
}

@Override
public void surfaceChanged(SurfaceHolder holder, int format, int width,
                           int height) {
}
```

GRAPHICS CANVAS BUBBLE SURFACEVIEW

```
@Override
public void surfaceCreated(SurfaceHolder holder) {
    mDrawingThread = new Thread(new Runnable() {
        public void run() {
            Canvas canvas = null;
            while (!Thread.currentThread().isInterrupted() && move()) {
                canvas = mSurfaceHolder.lockCanvas();
                if (null != canvas) {
                    drawBubble(canvas);
                    mSurfaceHolder.unlockCanvasAndPost(canvas);
                }
            }
        }
    });
    mDrawingThread.start();
}

@Override
public void surfaceDestroyed(SurfaceHolder holder) {
    if (null != mDrawingThread)
        mDrawingThread.interrupt();
}
```

VIEW ANIMATION

CHANGING THE PROPERTIES OF A VIEW OVER
A PERIOD OF TIME

SIZE

POSITION

TRANSPARENCY

ORIENTATION

VIEW ANIMATION CLASSES

TRANSITIONDRAWABLE

ANIMATIONDRAWABLE

ANIMATION

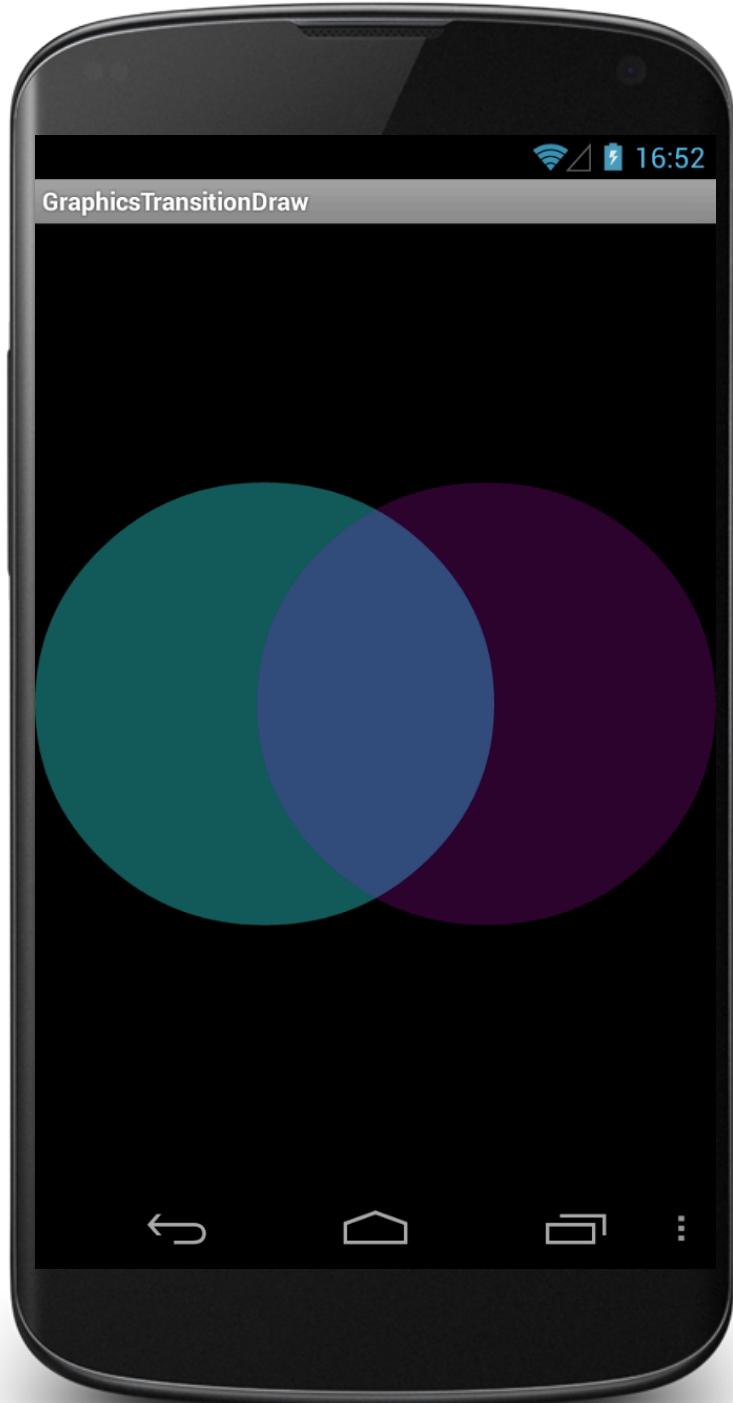
TRANSITIONDRAWABLE

A 2-LAYER DRAWABLE

CAN FADE BETWEEN 1ST & 2ND LAYERS

GRAPHICSTRANSITIONDRAWABLE

THIS APPLICATION USES THE SAME SHAPES AS
THE GRAPHICSSHAPEDRAW APPLICATIONS
SHOWS CYAN SHAPE THEN FADES TO
MAGENTA SHAPE



GRAPHICS TRANSITION DRAWABLE

```
<ImageView  
    android:id="@+id/image_view"  
    android:layout_width="match_parent"  
    android:layout_height="250dp"  
    android:contentDescription="@string/fading_image_desc"  
    android:layout_centerVertical="true"  
  
    . . .  
  
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
  
    setContentView(R.layout.main);  
  
    TransitionDrawable transition = (TransitionDrawable) getResources()  
        .getDrawable(R.drawable.shape_transition);  
  
    transition.setCrossFadeEnabled(true);  
  
    ((ImageView) findViewById(R.id.image_view)).setImageDrawable(transition);  
  
    transition.startTransition(5000);  
}
```

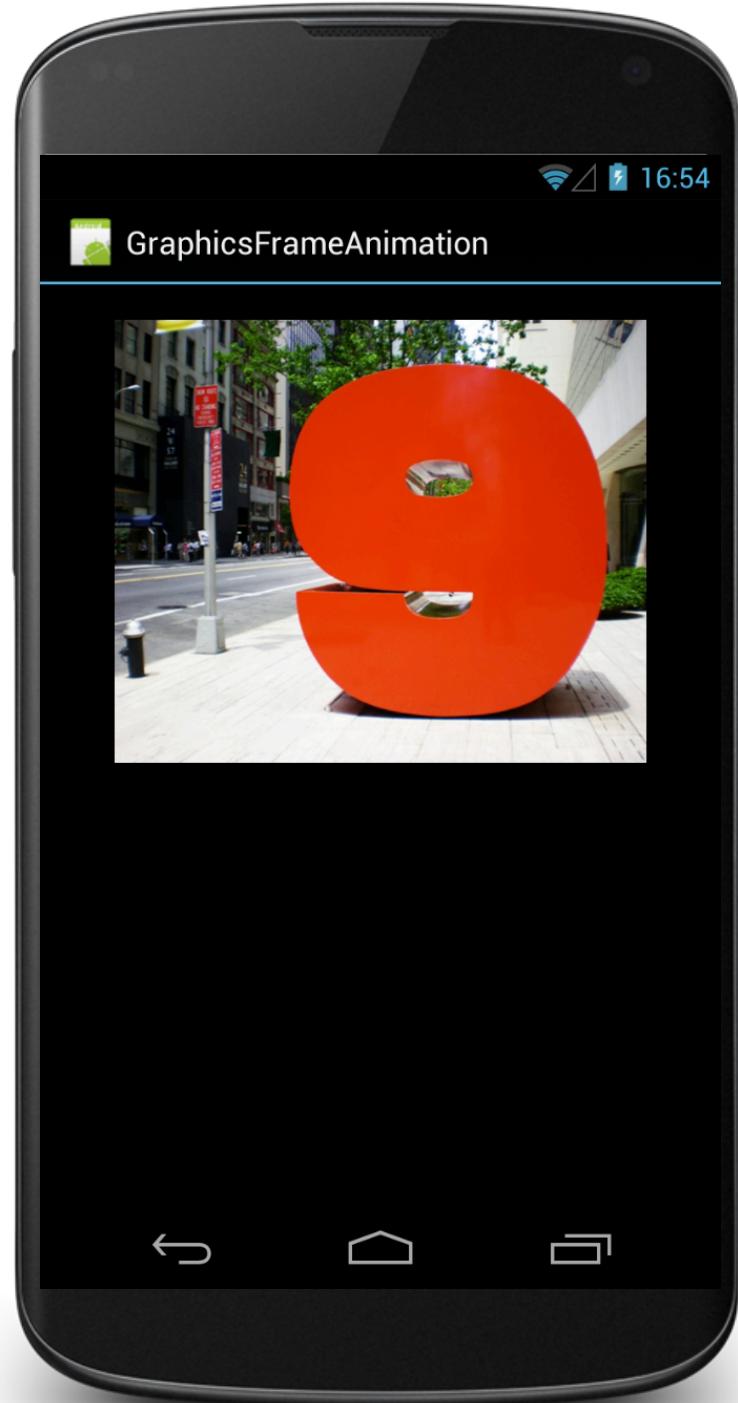
ANIMATIONDRAWABLE

ANIMATES A SERIES OF DRAWABLES

EACH DRAWABLE IS SHOWN FOR A SPECIFIC
AMOUNT OF TIME

GRAPHICS FRAME ANIMATION

USES AN ANIMATION DRAWABLE TO PRESENT
A FRAME BY FRAME ANIMATION



GRAPHICS FRAME ANIMATION

```
<ImageView  
    android:id="@+id/countdown_frame"  
    android:layout_width="300dp"  
    android:layout_height="250dip"  
    android:layout_gravity="center"  
    android:layout_marginBottom="20dp"  
    android:layout_marginTop="20dp"  
    android:contentDescription="@string/animation_desc"  
    android:scaleType="centerCrop" />
```

GRAPHICSFRAMEANIMATION

```
private AnimationDrawable mAnim;

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

    ImageView imageView = (ImageView) findViewById(R.id.countdown_frame);
    imageView.setBackgroundResource(R.drawable.view_animation);

    mAnim = (AnimationDrawable) imageView.getBackground();
}
```

GRAPHICS FRAME ANIMATION

```
@Override  
protected void onPause() {  
    super.onPause();  
    if (mAnim.isRunning()) {  
        mAnim.stop();  
    }  
}  
  
@Override  
public void onWindowFocusChanged(boolean hasFocus) {  
    super.onWindowFocusChanged(hasFocus);  
    if (hasFocus) {  
        mAnim.start();  
    }  
}
```

ANIMATION

A SERIES OF TRANSFORMATIONS APPLIED TO
THE CONTENT OF A VIEW

CAN MANIPULATE ANIMATION TIMING TO GIVE
EFFECT OF SEQUENTIAL OR SIMULTANEOUS
CHANGES

GRAPHICSTWEENANIMATION

APPLICATION DISPLAYS A SINGLE IMAGEVIEW
AND ANIMATES SEVERAL OF ITS PROPERTIES



GRAPHICSTWEENANIMATION

```
<ImageView  
    android:id="@+id/icon"  
    android:layout_width="200dp"  
    android:layout_height="200dp"  
    android:src="@drawable/b128"  
    android:contentDescription="@string/bubble_desc"  
    android:visibility="invisible"/>
```

GRAPHICSTWEENANIMATION

```
public class GraphicsTweenAnimationActivity extends Activity {  
  
    private ImageView mImageView;  
    private Animation mAnim;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
  
        mImageView = (ImageView) findViewById(R.id.icon);  
  
        mAnim = AnimationUtils.loadAnimation(this, R.anim.view_animation);  
    }  
}
```

GRAPHICSTWEENANIMATION

```
    @Override
    public void onWindowFocusChanged(boolean hasFocus) {
        super.onWindowFocusChanged(hasFocus);
        if (hasFocus) {
            mImageView.startAnimation(mAnim);
        }
    }
}
```

PROPERTY ANIMATION

ANIMATION – CHANGING PROPERTIES OF AN
OBJECT OVER A PERIOD OF TIME

PROPERTY ANIMATION ARCHITECTURE

VALUEANIMATOR – TIMING ENGINE

TIMEINTERPOLATOR – DEFINES HOW
VALUES CHANGE AS A FUNCTION OF TIME

ANIMATORUPDATERLISTENER – CALLED
BACK AT EVERY ANIMATION FRAME
CHANGE

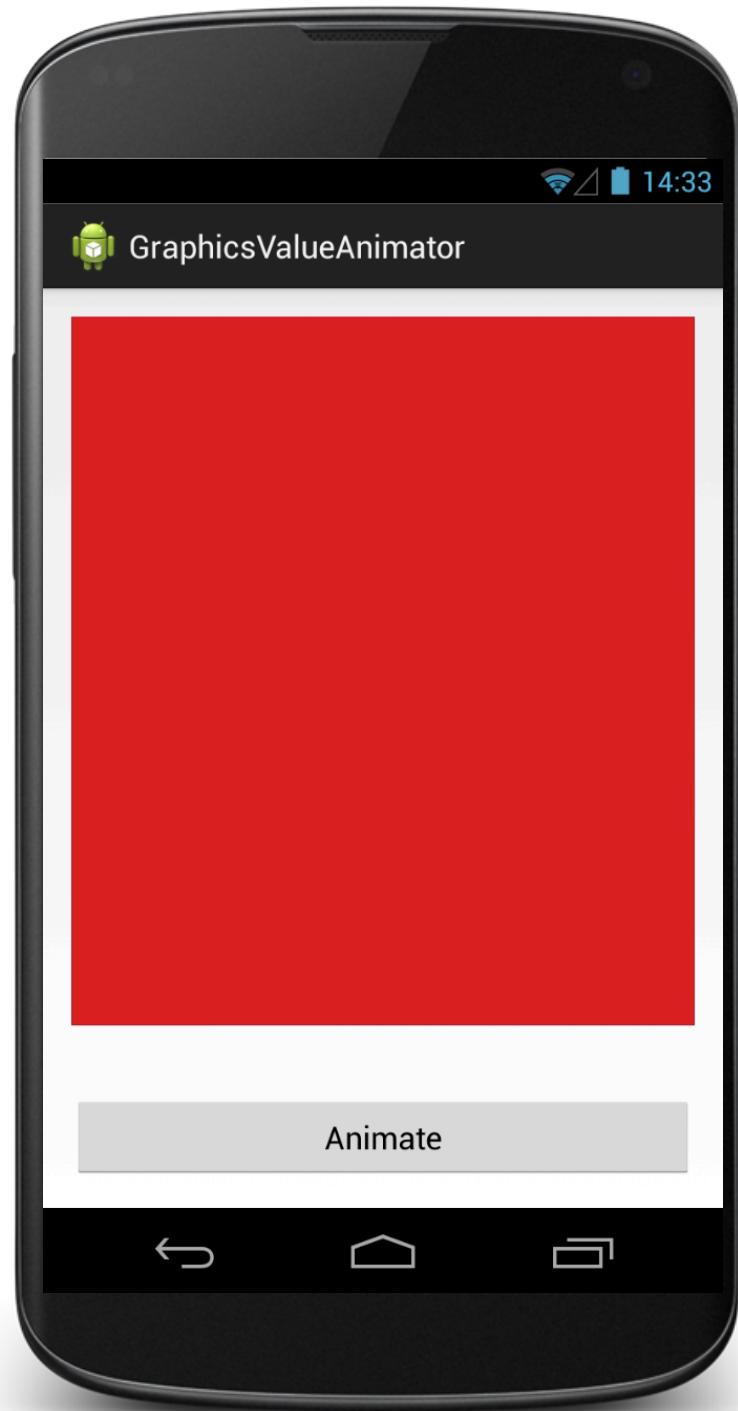
TYPEEVALUATOR – CALCULATES A
PROPERTY'S VALUE AT A GIVEN POINT IN
TIME

PROPERTY ANIMATION ARCHITECTURE

ANIMATORSET – COMBINES INDIVIDUAL
ANIMATIONS TO CREATE MORE COMPLEX
ANIMATIONS

GRAPHICSVALUEANIMATOR

USES A VALUEANIMATOR TO ANIMATE
CHANGING AN IMAGEVIEW'S BACKGROUND
COLOR



GRAPHICSVALUEANIMATOR

```
<ImageView  
    android:id="@+id/image_view"  
    android:layout_width="match_parent"  
    android:layout_height="400dp"  
    android:contentDescription="@string/app_name" />  
  
<Button  
    android:id="@+id/start_animation_button"  
    android:layout_width="match_parent"  
    android:layout_height="48dp"  
    android:layout_alignParentBottom="true"  
    android:text="@string/button_label"/>
```

GRAPHICSVALUEANIMATOR

```
public class ValueAnimatorActivity extends Activity {

    protected static final String TAG = "ValueAnimatorActivity";
    final private static int RED = Color.RED;
    final private static int BLUE = Color.BLUE;

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

        Button startButton = (Button) findViewById(R.id.start_animation_button);
        startButton.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                startAnimation();
            }
        });
    }
}
```

GRAPHICS VALUE ANIMATOR

```
public void startAnimation() {  
  
    final ImageView imageView = (ImageView) findViewById(R.id.image_view);  
  
    ValueAnimator anim = ValueAnimator.ofObject(new ArgbEvaluator(), RED,  
        BLUE);  
  
    anim.addUpdateListener(new AnimatorUpdateListener() {  
  
        @Override  
        public void onAnimationUpdate(ValueAnimator animation) {  
            imageView.setBackgroundColor((Integer) animation  
                .getAnimatedValue());  
        }  
    });  
  
    anim.setDuration(10000);  
    anim.start();  
}
```

GRAPHICSVIEWPROPERTYANIMATOR

SAME AS THE GRAPHICSTWEENANIMATION,
USES THE VIEWPROPERTYANIMATOR CLASS,
WHICH IS A SIMPLIFIED ANIMATOR FOR VIEWS



GRAPHICSVIEWPROPERTYANIMATOR

```
<ImageView  
    android:id="@+id/icon"  
    android:layout_width="200dp"  
    android:layout_height="200dp"  
    android:src="@drawable/b128"  
    android:alpha="0"  
    android:contentDescription="@string/bubble_desc"/>
```

GRAPHICSVIEWPROPERTYANIMATOR

```
public class GraphicsViewPropertyAnimatorActivity extends Activity {

    private ImageView mImageView;

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

    }

    @Override
    public void onWindowFocusChanged(boolean hasFocus) {
        super.onWindowFocusChanged(hasFocus);

        mImageView = (ImageView) findViewById(R.id.icon);

        if (hasFocus) {
            fadeIn.run();
        }
    }
}
```

GRAPHICSVIEWPROPERTYANIMATOR

```
Runnable fadeIn = new Runnable() {
    public void run() {
        mImageView.animate().setDuration(3000)
            .setInterpolator(new LinearInterpolator()).alpha(1.0f)
            withEndAction(rotate);
    }
};

Runnable rotate = new Runnable() {
    public void run() {
        mImageView.animate().setDuration(4000)
            .setInterpolator(new AccelerateInterpolator())
            .rotationBy(720.0f).withEndAction(translate);
    }
};

Runnable translate = new Runnable() {
    public void run() {
        float translation = getResources()
            .getDimension(R.dimen.translation);
        mImageView.animate().setDuration(3000)
            .setInterpolator(new OvershootInterpolator())
            .translationXBy(translation).translationYBy(translation)
            .withEndAction(scale);
    }
};
```

GRAPHICSVIEWPROPERTYANIMATOR

```
Runnable scale = new Runnable() {
    public void run() {
        mImageView.animate().setDuration(3000)
            .setInterpolator(new AnticipateInterpolator())
            .scaleXBy(1.0f).scaleYBy(1.0f).withEndAction(fadeOut);
    }
};

Runnable fadeOut = new Runnable() {
    public void run() {
        mImageView.animate().setDuration(2000)
            .setInterpolator(new DecelerateInterpolator()).alpha(0.0f);
    }
};
```

NEXT TIME

MULTITOUCH & GESTURES