Etch-a-Sketchish tutorial

Karl Thibaudeau

Steps:

1: Create drawable view with paint functions

2: Create main layout (include your custom view!)

3: Create buttons + functions

4: Program your main activity

5: Create brush size layout

Step 1:

Create a blank class, extend View. This will be your drawing canvas. In the class, create your global variables and implement an OnTouchListner.

```
public class DrawingView extends View implements View.OnTouchListener{
    private Path drawingPath;
    private Paint drawPaint;
    private Canvas drawingCanvas;
    private Bitmap bitmap;
    int CanvasWidth;
    int CanvasHeight;
    int CurrentPosX;
   int CurrentPosY;
   public static int paintColor;
   public static int brushSize;
    Context mainContext;
    View mainView;
```

Inside. In the basic method, have the parameters take in Context and AttributeSet. Attribute set is an object that take basic attributes for views from androids basic views (SetText ,setFocus, etc….).

```java
public DrawingView(Context context,AttributeSet attrs) {
    super(context,attrs);

    paintColor = Color.BLUE;
    brushSize = 50;
  mainContext = this.getContext();
    setUpDrawing();
}
```

Implement onSizeChanged method into the view. This will grab the height and width of our current view and allow variables to be set. In the method, we set the height and width of our canvas and set it into a bitmap (A bitmap is just the grid of pixels layer out of the screen and given logical addresses). The Bitmap.Config.ARGB 8888 sets the bitmap data type. ARGB 8888 sets how the data will be stored for each address in the bitmap (32 bits). Set your canvas (drawingCanvas) with the bitmap. Set your CurrentPosition variables for your X and Y lines. (This works like a grid!!). My example starts in the center of the canvas. Set your onTouchLisnter to this view.

```java
@Override
protected void onSizeChanged(int w, int h, int oldw, int oldh) {
//instasiate the size of the DrawingView layout item in the layout. This grabs the 
values which can
    //be assigned to our canvas item
    super.onSizeChanged(w, h, oldw, oldh);
    CanvasHeight = h;
    CanvasWidth = w;
    bitmap = Bitmap.createBitmap(w,h,Bitmap.Config.ARGB_8888);
    drawingCanvas = new Canvas(bitmap);
    //find center to start.
    CurrentPosY = Math.round(h/2);
    CurrentPosX = Math.round(w/2);

    this.setOnTouchListener(this);

}
```

Next, create your setup drawing method. Initialize your Path and Paint objects. Set your paint object colour to the paintColor variable (which you set to blue by default). SetAntiAlias provides a smoother paint with less “pixelated” look but is optional and makes little difference in this application. The stroke with is set to the Brush size which is 20 by default. Set style to Stroke, this is basic for paint objects that work with a path on a canvas. Stroke join defines how the path will join with other objects. Round will create a semi-circle point on the edges. The cap defines the shape of the brush. This can be Round or square by default, but anything is possible (even using images).

```
public void setUpDrawing(){
    drawingPath = new Path();
    drawPaint = new Paint();

    drawPaint.setColor(paintColor);

    drawPaint.setAntiAlias(true);
    drawPaint.setStrokeWidth(brushSize);
    drawPaint.setStyle(Paint.Style.STROKE);
    drawPaint.setStrokeJoin(Paint.Join.ROUND);
    drawPaint.setStrokeCap(Paint.Cap.ROUND);
}
```

Next, we will create our ontouch method that gets called whenever the view gets touched. This touch will set the point where we start a new path.

```
@Override
public boolean onTouch(View v, MotionEvent event) {
    if(event.getAction() == MotionEvent.ACTION_DOWN){

        CurrentPosY = Math.round(event.getY());
        CurrentPosX = Math.round(event.getX());
        Toast.makeText(mainContext, Integer.toString(CurrentPosY),
Toast.LENGTH_SHORT).show();

    }
    return false;
}
```

Next, on to the actual drawing. We will use a function that takes an integer to determine direction. We set the path pointer to the current position selected (on start is the center of the canvas). We then update the position to be relative to the brush size. In this case we use brushsize/6 since the path is a circle and we want each “stroke” to cover 1/6th of the last circle to create a smooth looking paint stroke. We then assign the path and paint to the canvas, which then draws to it’s bitmap. We then reset the path. After the switch statement we call invalidate and return true. This will call the onDraw() method below.

```
public boolean onSetAction(int Direction){

    switch(Direction){
        case 1:
            //Up

            drawingPath.moveTo(CurrentPosX,CurrentPosY);
            //set to how far you want each stroke to go.
            CurrentPosY = CurrentPosY - (brushSize/6);
           drawingPath.lineTo(CurrentPosX,CurrentPosY );
            drawingCanvas.drawPath(drawingPath,drawPaint);
            drawingPath.reset();
            break;
        case 2:
            //down

            drawingPath.moveTo(CurrentPosX,CurrentPosY);
            CurrentPosY = CurrentPosY + (brushSize/6);
            drawingPath.lineTo(CurrentPosX,CurrentPosY );
            drawingCanvas.drawPath(drawingPath,drawPaint);
            drawingPath.reset();
            break;
        case 3:
            //left



            drawingPath.moveTo(CurrentPosX,CurrentPosY);
            CurrentPosX = CurrentPosX -(brushSize/6);
            drawingPath.lineTo(CurrentPosX,CurrentPosY );
            drawingCanvas.drawPath(drawingPath,drawPaint);
            drawingPath.reset();
            break;
        case 4:
            //right

            drawingPath.moveTo(CurrentPosX,CurrentPosY);
            CurrentPosX = CurrentPosX +(brushSize/6);
            drawingPath.lineTo(CurrentPosX,CurrentPosY );
            drawingCanvas.drawPath(drawingPath,drawPaint);
            drawingPath.reset();
            break;
            default:
                return false;

    }
    invalidate();
    return true;

}
```

Now we create the overridden method OnDraw method. This method is built in to the View object and creates a new canvas item, which we then draw out path and paint to.

```
@Override
protected void onDraw(Canvas canvas) {
    canvas.drawBitmap(bitmap, 0, 0,drawPaint );
    canvas.drawPath(drawingPath, drawPaint);
}
```

Now, we create a method to clear the view. This is a simple method that erases the canvas, resets the start positions to the center of the screen and calls the onDraw method through invalidate to reset the canvas.

```java
public boolean clearCanvas(){

    bitmap.eraseColor(Color.WHITE);
    CurrentPosX = CanvasWidth/2;
    CurrentPosY= CanvasHeight/2;
 invalidate();
 return true;

}
```

Step 2: Create layout

Use this as your main activities layout. Notice the draw view that we created in there! Keep these set attributes.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="#ccc"
    tools:context="com.example.tkarl.etchasketchish.DrawingBoard"
    android:id="@+id/main_Layout">

    <TextView
        android:id="@+id/testtext"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/draw_text"
        android:textSize="25sp"
        android:gravity="center_vertical|center_horizontal"

       />
    <com.example.tkarl.etchasketchish.DrawingView
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/canvas_board"
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_marginBottom="3dp"
        android:layout_marginLeft="5dp"
        android:layout_marginRight="5dp"
        android:layout_marginTop="3dp"
        android:layout_weight="1"
        android:background="#FFFFFFFF"
        >

    </com.example.tkarl.etchasketchish.DrawingView>
<LinearLayout

    android:id="@+id/option_layout"
    android:layout_width="fill_parent"
    android:layout_height="50dp"
    android:orientation="horizontal"
    android:background="#fff"
    android:gravity="center_horizontal">

</LinearLayout>
    <LinearLayout
        android:id="@+id/Arrow_Holder"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:gravity="center_horizontal">
        <ImageButton
            android:id="@+id/Choose_Colour"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/ic_choose_colour"
            android:contentDescription="Colours"/>

        <ImageButton
            android:id="@+id/arrow_up"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/upward_arrow"
            android:contentDescription="Draw Up"
            />
        <ImageButton
            android:id="@+id/arrow_down"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/arrow_downwards"/>
        <ImageButton
            android:id="@+id/arrow_left"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/arrow_left"/>
        <ImageButton
            android:id="@+id/arrow_right"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/arrow_right"
           />
        <ImageButton
            android:id="@+id/Choose_Brush_Size"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"

            android:src="@drawable/ic_brush_size"
            android:contentDescription="Brush Size"/>
    </LinearLayout>
```

```
</LinearLayout>
```

Step 3: Create button images

Using the image asset tool, create images for your ImageButtons in your drawables directory. You can use the clipart feature to get all the images used in the demonstration.

Step 4: Program your main activity

Most the concepts in the main layout have been covered in class and should be simple to understand. Declare your buttons, set onClickListeners and send the direction to the drawview buy calling the drawview as you would call any other view to get access to it’s public functions and variables.

```
@Override
public void onClick(View v) {
        switch (v.getId()){
            case R.id.arrow_up:
                drawView.onSetAction(1);
                break;
            case R.id.arrow_down:
                drawView.onSetAction(2);
                break;
            case R.id.arrow_left:
                drawView.onSetAction(3);
                break;
            case R.id.arrow_right:
                drawView.onSetAction(4);
                break;
         }
}
```

To set the colors or brush size, you can inflate a layout like I did and use it’s buttons to change the values in the drawview or do it through settings. Your choice.

Step 5: Clear the canvas on shake

Using the accelerometer sensor, we can detect the motion of the phone. Using simple math we can detect the speed of the accelerometer in different directions, hence detecting if the device has been shaken. Since the accelerometer works by detection motion in 3 dimensions (6 directions), we can detect whether the phone has been shaken vertically. Basically, the upwards motion of the phone (y) – the downwards motion of the phone (-y) divided by the time in between motions (in seconds) detects the speed. We set the speed result to an absolute value. If the speed reaches a certain threshold (I found 800 works) then we call our drawview's clearCanvas method.

```
@Override
public void onSensorChanged(SensorEvent event) {
    Sensor accel = event.sensor;
    if (accel.getType() == Sensor.TYPE_ACCELEROMETER){
        float y = event.values[1];
        long curTime = System.currentTimeMillis();
        if ((curTime - lastupdate) > 200) {
            long diffTime = (curTime - lastupdate);
            lastupdate = curTime;

            float speed = Math.abs( y  - lasty)/ diffTime * 10000;

            if (speed > 800) {
              drawView.clearCanvas();
            }


            lasty = y;

        }
    }
}
```

THE END! Thanks for reading and happy coding!