Android 2-D Drawing

Android uses a Canvas object to host its 2-D drawing methods. The program below draws a blue circle on a white canvas. It does not make use of the main.xml layout but draws directly on the Activity

```
package android.com.drawing;
import android.app.Activity;
import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.os.Bundle;
import android.view.View;

public class DrawSpace extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        DrawHere d = new DrawHere(this);
        //the setContentView is changed to not use main.xml
        setContentView(d);
    }
    //inner class - no public
    class DrawHere extends View{
     //constructor for this class
       public DrawHere(Context context) {
             super(context);
         }
         @Override
         public void onDraw(Canvas c) {
               Paint paint = new Paint();
               paint.setStyle(Paint.Style.FILL);
               paint.setColor(Color.WHITE);
               c.drawPaint(paint);
               paint.setAntiAlias(true);
               paint.setColor(Color.BLUE);
               c.drawCircle(80, 20, 15, paint);
         }
    }
}
```

Note that there are 2 classes defined here, DrawSpace was created with the Android Activity project, it could have been assigned any reasonable name but needs to extend Activity. DrawHere is a class within the { }s of DrawSpace and it extends View, again any reasonable name can be assigned. The constructor of the class must be present with the same name as its class.

In Android a class can only directly inherit from one other class so the default Activity DrawSpace cannot extend Activity and View.

The Canvas class is required as an area (canvas) to draw objects on. The Paint class provides a range of colours and tools to draw with.

The following are common drawing methods within the Canvas class.

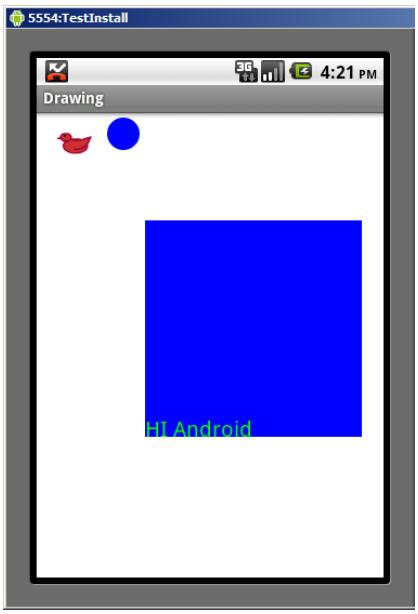
```
drawCircle(centreX,centreY,radius,paint);
drawLine(startX,startY,stopX, stopY,paint);
drawRect(left,top,right,bottom);
drawText(text,startX,startY,paint);
```

This shows them in use to create a few shapes:

```
paint.setColor(Color.BLUE);
c.drawCircle(80, 20, 15, paint);
c.drawRect(100, 100, 300, 300, paint);
paint.setTextSize(20);
paint.setColor(Color.GREEN);
c.drawText("HI Android",  100, 300, paint);
```

An image can be drawn if an appropriate png file is in /res/drawable-hdpi. The numbers 10,10 are the x,y coordinates where the bitmap is drawn.

```
Bitmap piccie = BitmapFactory.decodeResource(getResources(),
R.drawable.bird);
c.drawBitmap(piccie, 10, 10, null);
```

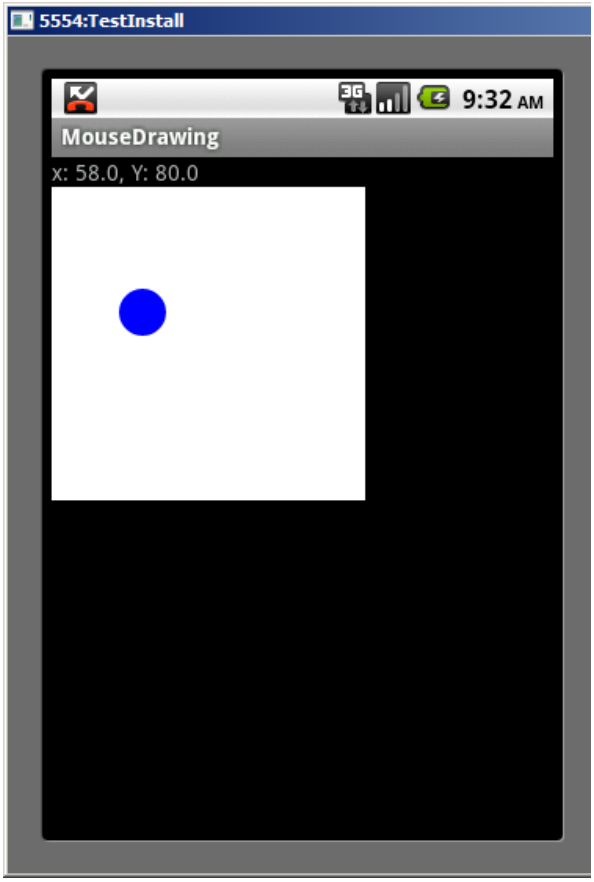


Using onTouch with a FrameLayout

The above code draws on the entire View. One Widget that is appropriate for hosting drawn Graphics is the FrameLayout, basically a box on the Activity screen. The obvious choice, ImageView is not a good start as it does not support all the methods we need for Graphics drawing.

The xml code below will add a FrameLayout of the specified dimensions with a white background. The FrameLayout can, probably should, be placed within another Layout Widget such as a LinearLayout.

```
<FrameLayout
        android:id="@+id/fLayout"
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:background="#FFFFFF">
</FrameLayout>
```



In this Activity a FrameLayout and TextView are present. The TextView `lblInfo` shows the coordinates from the onTouch event of the FrameView.

The code below will add a Listener to the FrameLayout and output the x and y from within that control.

In programming the origin of (x,y) is at the top left of the object.

Note that the code that begins with `setOnTouchListener` is essentially 1 line so ends with closing brackets and a semi colon.

```
flView = (FrameLayout)findViewById(R.id.fLayout);
flView.setOnTouchListener(new View.OnTouchListener()
          @Override
          public boolean onTouch(View v, MotionEvent event) {
               String coord = "x: " +
               String.valueOf(event.getX());
               coord = coord + ", Y: " +
               String.valueOf(event.getY());
               lblInfo.setText(coord);
               return false;
          }

});
```

Drawing on the FrameLayout

The same problem is going to crop up as before, we need a View and an Activity but a single class cannot extend both. This is solved by a new inner class that extends View and has an onDraw() method that creates a Canvas.

```
class DrawHere extends View{
      private final float x;
      private final float y;
     //constructor for this class
      public DrawHere(Context context, float xPos, float yPos)
     {
              super(context);
              this.x = xPos;
              this.y = yPos;

       }
       @Override
       public void onDraw(Canvas c) {
                Paint paint = new Paint();
                paint.setStyle(Paint.Style.FILL);
                paint.setColor(Color.WHITE);
                c.drawPaint(paint);
                paint.setAntiAlias(true);
                paint.setColor(Color.BLUE);
                c.drawCircle(x, y, 15, paint);
       }
}
```

This class aims to draw a blue circle on a white background at the x, y values passed to it by the constructor. A circle also needs a radius value passing to it. Here a value of 15 has been used.

The onTouch method is expanded to create a new Object of the Class DrawHere and position it at the x and y coordinates picked up by onTouch.

```
float x = event.getX();
float y = event.getY();
flView.addView
(new DrawHere(findViewById(R.id.fLayout).getContext(),x,y));
```

flView is the Java object linked to the XML object *fLayout*.

Drawing an existing image

The above code draws a blue circle. The drawing methods of the canvas object offer a limited range of 2-D shapes to draw.

An alternative is to draw an existing Bitmap object. A Bitmap object needs to be declared global to the class. This can be instantiated in onCreate so the image only needs to be loaded once.

```
if(picBack == null)
     picBack = BitmapFactory.decodeResource(getResources(),
R.drawable.ic_launcher);
```

This will load the default Android launch icon although another images within res/drawable-hdpi could be pointed to.

The drawing code in onDraw will now be:

```
c.drawBitmap(picBack, x, y, null);
```

Browsing for an image

One source of possible images is the picture gallery of the device. The easy way to get an image onto the device and make it available is to browse for a website, find an image and long click on it to download the image.

If the virtual device will not access the web there is another way. The program adb Android Debug Bridge) is installed within Android tools in the Android SDK directory. This program runs from the command line and can push and pull files from the virtual device.

The code below (running from Windows) moves a jpg file to the `/sdcard/DCIM` directory of the device from the folder that adb is running from.

```
C:\Program Files\Android\android-sdk\platform-tools>adb push
Penguins.jpeg /sdcard/DCIM/penguins.jpeg
```

The adb program can also open a shell on the remote device as a check that the file has been created.

```
C:\Program Files\Android\android-sdk\platform-tools>adb shell


# ls /sdcard/*

ls /sdcard/*

100ANDRO

penguins.jpeg

images.jpeg
```

Any new files that are transferred will not show up until the virtual machine is closed down and reloaded.

Loading the image

Here the code to launch the gallery has been linked to a click event listening on a Button.

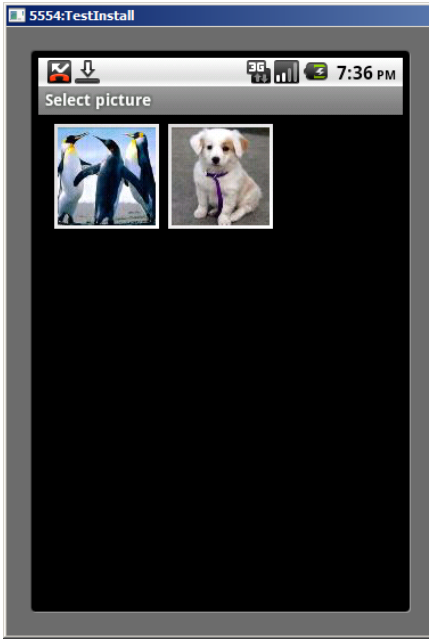
```
public void onClick(View v) {
  if(v== findViewById(R.id.bLoad))
  {
     Intent pickPhoto = new Intent(Intent.ACTION_PICK);
     pickPhoto.setType("image/*");
     startActivityForResult(pickPhoto,1);
  }
}
```

This will open the gallery as an Intent in the same way that an additional Activity might be opened. The following method is needed to process the result of picking an image.

```
protected void onActivityResult(int requestCode, int
resultCode, Intent data) {
     switch(requestCode) {
     case 1:
          if(resultCode == Activity.RESULT_OK)
          {
               Uri imagePic = data.getData();
               try {
//swap out from the default drawing bitmap used above
     picBack = Media.getBitmap(getContentResolver(),imagePic);
//scale as required for drawing the image
```

```
     picBack = Bitmap.createScaledBitmap(picBack,
(int)(picBack.getWidth() / 4),(int)(picBack.getHeight() / 4),
false);
                    }
                    catch (FileNotFoundException e) {
                         e.printStackTrace();
                    } catch (IOException e) {
                         e.printStackTrace();
                    }
               }//end if
          }//end switch
}//end method
```

The try and catch blocks are required for file handling here but Eclipse should fill them in.



In this example an ImageView has been placed below the FrameLayout to test the image loading