ANDROID

第３回Androidセミナーin大阪

Canvas使ってスタンプポンポン!

エモーションプラス　原　秀樹

...Let's keep happy by more achievement feelings.

EMOTIONPLUS

自己紹介

- 実は「現役の教師」、そして「フリープログラマ」
  - 専門学校HAL大阪で常任の教官としてゲーム、制御・ロボット系の分野を幅広く指導
  - 一方、個人事業（屋号：エモーションプラス）としてソフトウェア開発関連事業を手掛けている。

- 得意分野はゲームソフト開発、マイコン制御ソフト開発など
  - 教師になる前はゲームメーカー数社でハードウェア制御ソフト開発業務やゲーム系ソフトウェア開発業務などを手掛ける。
  - 過去にPlayStation用ソフトやアーケードゲームを開発したこともある。
    （全然売れんかったけどw）

- パソコン歴はもはや30年近く！
  - 最初に触れたマシンはNEC PC-8001やシャープMZ-80B
  - C言語もいいけど「やっぱアセンブラって気持ちええわ～！」と思うタイプ

どうやって作ろうか？

- 画面にスタンプを押すアプリを作りたい！
  - タッチパネルを**指でタッチしたタイミングで座標を取得**しよう
  - 取得した座標を使って**スタンプのパターンを描画**しよう

- じゃ、**Androidでどうやって実現すればいい？**
  - タッチパネルへのタッチと座標取得は…
    - onTouchEventでいけそう！
  - 画面への描画方法は…
    - Canvasクラスとか Bitmapクラスを使えば大丈夫！
  - 用意してあるビットマップを使った描画方法は…
    - drawableリソースを使ってOK！

タッチパネルのイベント

- タッチパネルがタッチされるとonTouchEventが発生し、座標値などを取得できます。

```
public boolean onTouchEvent(MotionEvent event) {
    switch (event.getAction()) {
    case MotionEvent.ACTION_DOWN:     // タッチした時
        mTouchX = (int)event.getX(); // X座標を取得
        mTouchY = (int)event.getY(); // Y座標を取得
        break;
    case MotionEvent.ACTION_UP:       // 指を離した時
        break;
    case MotionEvent.ACTION_MOVE:     // タッチしつつ指を動かした時
        break;
    }
    return super.onTouchEvent(event);
}
```

描画で使うリソース

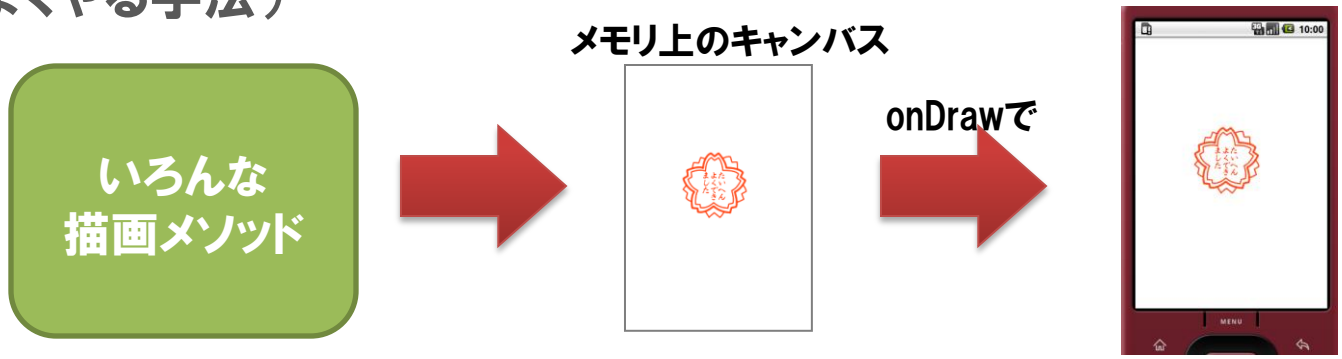
- スタンプのパターンはdrawableリソースとして扱います。
- **事前に画像ファイルを準備しましょう!!**
  - Androidでは画像形式として**「PNG」を推奨**（JPEG,GIFもOK）。
  - 今回は**縦100ドット、横100ドット程度のサイズでOK**です。

このような画像があればいいですが、
ペイントソフトでちゃちゃっと落書きみたいに
描いた画像でも全然OKですよ！

- 画像ファイルはプロジェクト内の res/drawableフォルダに入れると、「R.drawable.ファイル名」というリソースIDで扱えるようになります。
  - （例）画像ファイル stamp.png をres/drawableフォルダに入れて、R.drawable.stampというIDで扱う

描画させるキャンバスを用意

- 描画するためには「キャンバス」を用意します。
- onDrawイベント（描画実行）で与えられるCanvasを、そのまま使っても別にいいけど…
- 今回はメモリ上にあらかじめ別にキャンバスを作っておいて、画像の描画はメモリ上キャンバスへ、onDraw時はそのキャンバスを本来のキャンバスに丸ごと貼り付ける方法を採用！（いわゆるオフスクリーン…ゲームソフトではよくやる手法）



キャンバス生成・描画の例

●（例）メモリ上にキャンバスを作成

```
private Bitmap mBitmap;     // ビットマップオブジェクト
private Canvas mCanvas;     // キャンバスオブジェクト（オフスクリーン）
// 描画用ビットマップを生成
mBitmap = Bitmap.createBitmap(320, 480, Bitmap.Config.RGB_565);
// オフスクリーンのキャンバスを生成（ビットマップを割り付け）
mCanvas = new Canvas(mBitmap);
// キャンバスを一度真っ白に塗りつぶす
mCanvas.drawColor(Color.WHITE);
```

●（例）onDraw時にキャンバスを丸ごと貼り付ける

```
protected void onDraw(Canvas canvas) {
    // オフスクリーンビットマップを描画（＝可視化）
    canvas.drawBitmap(mBitmap, 0, 0, null);
}
```

drawableリソースを使った描画

- まずメモリに読み込んでビットマップとして扱えるよう変換しておきます。

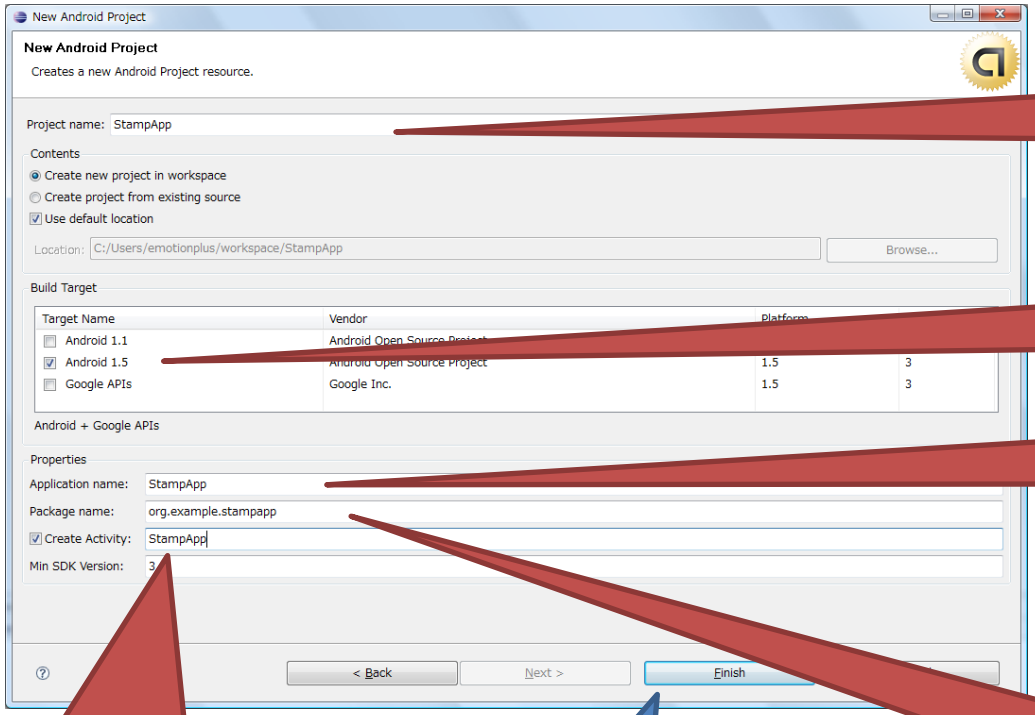
```
private Bitmap  mStamp;     // スタンプ用ビットマップデータ
// drawableリソースからスタンプ用ビットマップを生成
mStamp = BitmapFactory.decodeResource(context.getResources(),
              R.drawable.stamp);
```

- 描画したい時にメモリ上キャンバスに対して描画メソッドを実行します。

```
mCanvas.drawBitmap(mStamp,
                   mTouchX - mStamp.getWidth()  / 2,
                   mTouchY - mStamp.getHeight() / 2,
                   null);
```

では早速作ってみましょう！

- まずはプロジェクトを作成します。



Project name:
StampApp

Android 1.5 に
チェックを入れる

Application name:
StampApp

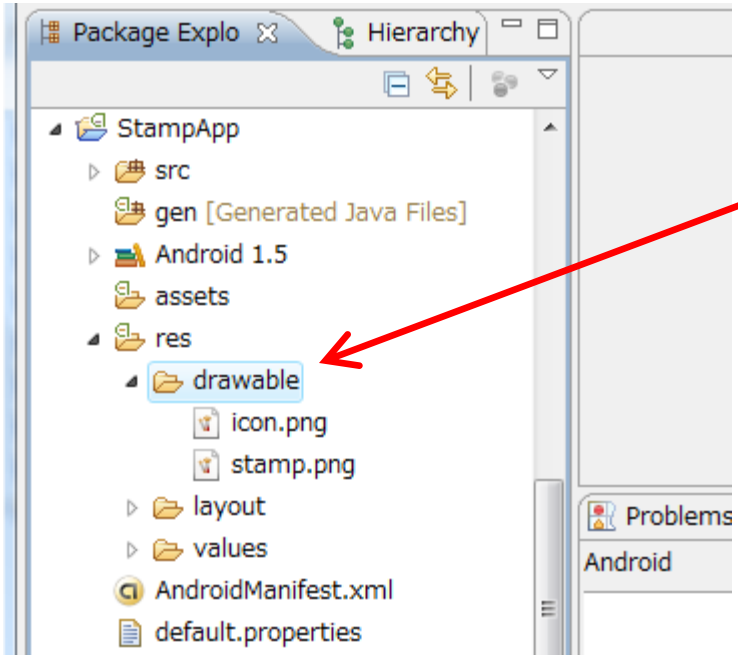
Create Activity:
StampApp

入力したら
Finishを…

Package name:
org.example.stampapp

リソースの準備

- プロジェクトのフォルダを見てみましょう。
  res/drawableフォルダに用意しておいた画像ファイルを入れてください。

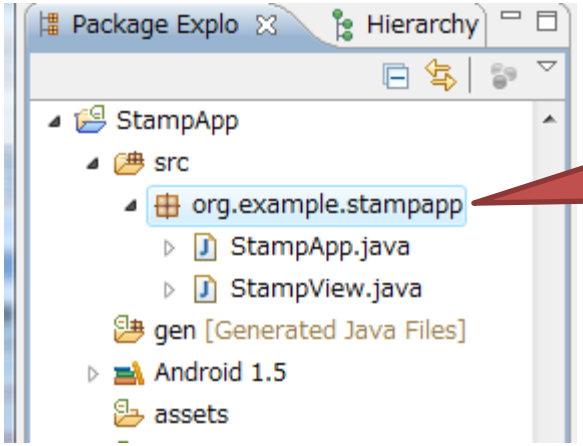


Stamp.png

画像ファイルは
res/drawable フォルダに
直接コピー＆ペーストできます

ソースファイルの準備

- 今回はタッチパネルイベントおよび一連の描画処理をViewクラスから派生させた「StampViewクラス」で行うので、**StampView.javaファイルを新規作成**します。
  - Viewクラスとは何ぞや？と思われる方もいらっしゃるかと思いますが、画面に関するクラスだと思ってもらえばとりあえずはいいかと…



ここで右クリックして
[New] > [File]を選択して
StampView.java
と入力してファイルを生成

Activityファイルの修正

- 自動で生成されているStampApp.javaを以下のように修正します。

```
package org.example.stampapp;
import android.app.Activity;
import android.os.Bundle;
import android.view.Window;

public class StampApp extends Activit
    /** Called when the activity is rst created. */
    @Override
    public void onCreate(Bundle avedInstanceState) {
        super.onCreate(savedInstanceState);
//      setContentView(R.layout.main);

        requestWindowFeature(Window.FEATURE_NO_TITLE);     // タイトルを消す
        setContentView(new StampView(this));     // StampViewをセット
    }
}
```

この行は削除!!

この行は追加!!

StampView.javaの作成（1／4）

- 新規作成で追加したStampView.javaに以下の内容を入力します。

```
package org.example.stampapp;

import org.example.stampapp.R;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.view.MotionEvent;
import android.view.View;

/*
 * StampViewクラス
 */
public class StampView extends View {
```

次のページへ続く

StampView.javaの作成（2/4）

```
private Bitmap  mBitmap;    // ビットマップオブジェクト
private Canvas  mCanvas;    // キャンバスオブジェクト（オフスクリーン）
private int     mTouchX;    // タッチされた時のX座標
private int     mTouchY;    // タッチされた時のY座標
private Bitmap  mStamp;     // スタンプ用ビットマップデータ

public StampView(Context context) {
    super(context);

    // 描画用ビットマップを生成
    mBitmap = Bitmap.createBitmap(320, 480, Bitmap.Config.RGB_565);
    // オフスクリーンのキャンバスを生成（ビットマップを割り付け）
    mCanvas = new Canvas(mBitmap);
    // キャンバスを塗りつぶす
    mCanvas.drawColor(Color.WHITE);
    // drawableリソースからスタンプ用ビットマップを生成
    mStamp = BitmapFactory.decodeResource(context.getResources(),
                R.drawable.stamp);
}
```

次のページへ続く

StampView.javaの作成（3／4）

```
/*
 * 描画時
 */
@Override
protected void onDraw(Canvas canvas) {
    // オフスクリーンビットマップを描画（＝可視化）
    canvas.drawBitmap(mBitmap, 0, 0, null);
}

/*
 * タッチイベントハンドラ
 */
@Override
public boolean onTouchEvent(MotionEvent event) {
    switch (event.getAction()) {
    case MotionEvent.ACTION_DOWN:
        // タッチされている座標を取得
        mTouchX = (int)event.getX();
        mTouchY = (int)event.getY();
```

次のページへ続く

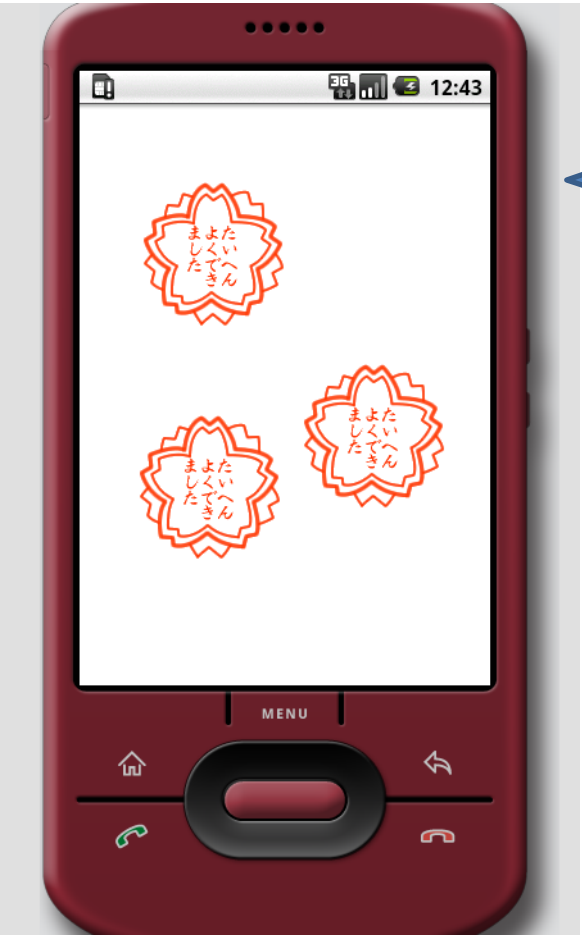
StampView.javaの作成（4/4）

```
                //イメージの描画
                mCanvas.drawBitmap(mStamp,
                                mTouchX - mStamp.getWidth() / 2,
                                mTouchY - mStamp.getHeight() / 2,
                                null);
                // 画面を無効化（＝再描画）
                invalidate();
            }
            return super.onTouchEvent(event);
        }
}
```

- 以上で終了です!!

  早速ビルドして動かしてみましょう!!

Emulator実行時の画面



指でタッチしたら
その場所に画像がポンと
描画されるはずです。

うまく動きましたか？

では、これで終了です。
お疲れさまでした!!
そして、有難うございました!!