android.graphics.Drawable

[Класс Drawable](http://developer.alexanderklimov.ru/android/catshop/android.graphics.drawable.php#drawable)  
[Класс BitmapDrawable](http://developer.alexanderklimov.ru/android/catshop/android.graphics.drawable.php#bitmapdrawable)  
[Класс ColorDrawable](http://developer.alexanderklimov.ru/android/catshop/android.graphics.drawable.php#colordrawable)  
[Класс GradientDrawable](http://developer.alexanderklimov.ru/android/catshop/android.graphics.drawable.php#gradientdrawable)  
[Класс TransitionDrawable](http://developer.alexanderklimov.ru/android/catshop/android.graphics.drawable.php#transitiondrawable)  
[Класс ShapeDrawable](http://developer.alexanderklimov.ru/android/catshop/android.graphics.drawable.php#shapedrawable)  
  [Класс RectShape](http://developer.alexanderklimov.ru/android/catshop/android.graphics.drawable.php#rectshape)  
  [Класс OvalShape](http://developer.alexanderklimov.ru/android/catshop/android.graphics.drawable.php#ovalshape)  
  [Класс RoundRect](http://developer.alexanderklimov.ru/android/catshop/android.graphics.drawable.php#roundrect)  
  [Класс Path](http://developer.alexanderklimov.ru/android/catshop/android.graphics.drawable.php#path)  
  [Класс ArcShape](http://developer.alexanderklimov.ru/android/catshop/android.graphics.drawable.php#arcshape)  
[ScaleDrawable](http://developer.alexanderklimov.ru/android/catshop/android.graphics.drawable.php#scaledrawable)  
[RotateDrawable](http://developer.alexanderklimov.ru/android/catshop/android.graphics.drawable.php#rotatedrawable)  
[LayerDrawable](http://developer.alexanderklimov.ru/android/catshop/android.graphics.drawable.php#layerdrawable)  
[StateListDrawable](http://developer.alexanderklimov.ru/android/catshop/android.graphics.drawable.php#statelistdrawable)  
[LevelListDrawable](http://developer.alexanderklimov.ru/android/catshop/android.graphics.drawable.php#levellistdrawable)

Класс Drawable

Android предлагает двумерную графическую библиотеку рисования **android.graphics.Drawable**. Класс **Drawable** является базовым классом для всех классов работы с графикой. Это общая абстракция для рисуемого объекта на холсте **Canvas**. Класс **Drawable** определяет разнообразные виды графики, включая:

* [AnimationDrawable](http://developer.alexanderklimov.ru/android/animation/frameanimation.php) - покадровая анимация. XML-аналог <animation-list>
* AnimatedRotateDrawable - анимация вращения. XML-аналог **<animated-rotate>**
* [AnimatedVectorDrawable](http://developer.alexanderklimov.ru/android/catshop/animatedvectordrawable.php) - анимация векторных объектов (Android 21). XML-аналог **<animated-vector>**
* BitmapDrawable - Bitmap - изображение PNG и JPG, создание, растягивание, выравнивание растра. XML-аналог <bitmap>
* ClipDrawable - XML-аналог <clip>
* ColorDrawable - заполнение цветом. XML-аналог <color>
* GradientDrawable - создание градиентов
* InsetDrawable - XML-аналог <inset>
* LayerDrawable - многослойные картинки, контейнеры для дочерних элементов визуализации. XML-аналог <layer-list>
* LevelListDrawable - контейнер для одного объекта-потомка. Основана на целом числе, например, индикатор заряда или уровень силы сигнала. XML-аналог <level-list>
* NinePathDrawable - масштабируемое PNG-изображения NinePath. Используется в качестве фонового изображения для кнопок с изменяемыми размерами. XML-аналог <nine-patch>
* PictureDrawable
* [RippleDrawable](http://developer.alexanderklimov.ru/android/catshop/rippledrawable.php) - анимационный эффект ряби (Android 21). XML-аналог <ripple>
* RotateDrawable - XML-аналог <rotate>
* [RoundedBitmapDrawable](http://developer.alexanderklimov.ru/android/catshop/roundedbitmapdrawable.php) - создание круглых изображений
* ScaleDrawable - контейнер для одного объекта-потомка. Изменяет размер на основе текущего заданного значения. Используется для создания вьювера картинок с возможностью увеличения. XML-аналог <scale>
* ShapeDrawable - Shape - команды рисования векторных объектов, основанные на Path. XML-аналог <shape>
* StateListDrawable - специальный контейнер. Используют для создания различных вариантов выделения кнопок и установки фокуса на них. XML-аналог <selector>
* TransitionDrawable - плавные переходы и др. XML-аналог <transition>
* [VectorDrawable](http://developer.alexanderklimov.ru/android/catshop/vectordrawable.php) - векторные изображения (Android 21). Аналог **<vector>**.

Если вы придумаете свой XML-тег типа <kitty> или сделаете опечатку, то получите исключение **Resources.NotFoundException**.

Есть два способа определить и инициализировать объект **Drawable**:

* использовать готовые изображения, сохраненные в каталоге **res/drawable/**
* использовать ХМL-файл, в котором будут определены свойства объекта **Drawable**

Самый простой способ добавить графику в приложение — это использование готовых изображений из ресурсов. Наиболее предпочтительный тип PNG, также можно использовать JPG (приемлемо) и GIF (нежелательно). Обычно таким способом загружают значки и картинки.

Ресурсы изображений, помещенные в каталог **res/drawable/** во время компиляции проекта, могут быть автоматически оптимизированы со сжатием изображения утилитой **aapt**. Если требуется загружать растровые изображения BMP, которые утилита **aapt** обязательно оптимизирует, без сжатия, поместите их в каталог **res/raw/**, где они не будут оптимизироваться. Однако потребуется загрузка с использованием методов потокового ввода-вывода и последующая конвертация графики в растровый рисунок.

В отличие от **View** объект **Drawable** не нуждается в разметке и размерах контейнера. Его задача - просто рисовать.

Размеры самого объекта **Drawable** можно узнать с помощью методов **getIntrinsicWidth()** и **getIntrinsicHeight()**. Также есть метод **getBounds()**, возвращающий ограничивающий прямоугольник.

С помощью метода **setState()** можно управлять состоянием рисунка, например, менять цвет при касании экрана. Метод **setLevel()** может задать уровень вывода картинки.

Объект **Drawable** можно создать с помощью XML в виде ресурсов и сохранить в папках со словом **drawable**. А затем присвоить, например объекту**ImageView** с помощью атрибута **android:src="@drawable/ic\_launcher"**.

Программно такого же эффекта можно достичь через код.

imageView.setImageResource(R.drawable.ic\_launcher);

Создание объектов Drawable из ресурсов

В других случаях вы можете захотеть обработать ваш ресурс изображения как объект **Drawable**. Чтобы это сделать, создайте **Drawable**, загрузив изображение из ресурсов примерно так:

Resources res = mContext.getResources();

Drawable drawable = res.getDrawable(R.drawable.ic\_launcher);

Каждый уникальный ресурс в вашем проекте может поддерживать только одно состояние независимо от того, сколько различных объектов вы можете инициализировать в программном коде для этого ресурса. Например, если вы инициализируете два объекта **Drawable** от одного ресурса изображения, а затем измените свойство, например **alpha** (прозрачность), для одного из объектов **Drawable**, другой объект также изменит это свойство.

Напишем простой пример и попробуем получить объект от одного и того же значка.

Resources res = getResources();

Drawable drawable1 = res.getDrawable(R.drawable.ic\_launcher);

Drawable drawable2 = res.getDrawable(R.drawable.ic\_launcher);

Log.i("Drawable", "drawable1: " + drawable1);

Log.i("Drawable", "drawable2: " + drawable2);

Как ни странно, но результаты различаются. У меня вышло следующее:

drawable1: android.graphics.drawable.BitmapDrawable@b1080a68

drawable2: android.graphics.drawable.BitmapDrawable@b1080ad8

Метод **getDrawable()** **всегда** возвращает новый **Drawable**, даже если источник один и тот же. Тем не менее метод **getConstantState()** вернёт одинаковый результат.

Log.i("Drawable", "drawable1: " + drawable1.getConstantState());

Log.i("Drawable", "drawable2: " + drawable2.getConstantState());

Смотрим логи:

drawable1: android.graphics.drawable.BitmapDrawable$BitmapState@b107e3b8

drawable2: android.graphics.drawable.BitmapDrawable$BitmapState@b107e3b8

C помощью **Drawable** можно [замостить экран активности](http://developer.alexanderklimov.ru/android/design/tilemode.php#tilemode), используя маленький **Bitmap**.

Конвертируем Drawable в Bitmap и выводим в ImageView

Предположим, у нас в ресурсах есть картинка (Drawable). Нам нужно конвертировать её в объект **Bitmap** и вывести в **ImageView**:

private ImageView mTargetImageView;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

mTargetImageView = (ImageView)findViewById(R.id.imageViewTarget);

// Загружаем растр из ресурсов

Bitmap localBitmap = BitmapFactory.decodeResource(

getApplicationContext().getResources(),

R.drawable.ic\_launcher);

mTargetImageView.setImageBitmap(localBitmap);

}

Программное создание Drawable

**Drawable** является абстрактным классом. Если вы собираетесь создавать класс на его основе, то необходимо реализовать следующие методы:

* **draw(Canvas canvas)** – рисовать на холсте
* **getOpacity()** - вернуть значение прозрачности Drawable. Используются константы:
  1. **UNKNOWN** – прозрачность неизвестна
  2. **TRANSPARENT** – Drawable будет полностью прозрачным
  3. **TRANSLUCENT** – Drawable будет состоять из прозрачных и непрозрачных участков
  4. **OPAQUE** – Drawable будет полностью непрозрачным
* **setAlpha(int alpha)** – установить прозрачность
* **setColorFilter(ColorFilter colorFilter)** – установить цвет при помощи фильтра **ColorFilter**

Создадим свой вариант **Drawable**, рисующий шестиугольник ([Источник для примера](http://startandroid.ru/ru/uroki/vse-uroki-spiskom/394-urok-166-risovanie-sozdanie-svoego-drawable.html)).

package ru.alexanderklimov.emptyactivity;

import android.graphics.Canvas;

import android.graphics.ColorFilter;

import android.graphics.Paint;

import android.graphics.Path;

import android.graphics.PixelFormat;

import android.graphics.Rect;

import android.graphics.drawable.Drawable;

public class HexagonDrawable extends Drawable {

private Paint mPaint = new Paint(Paint.ANTI\_ALIAS\_FLAG);

private Path mPath = new Path();

@Override

public void draw(Canvas canvas) {

canvas.drawPath(mPath, mPaint);

}

@Override

public void setAlpha(int alpha) {

mPaint.setAlpha(alpha);

}

@Override

public void setColorFilter(ColorFilter colorFilter) {

mPaint.setColorFilter(colorFilter);

}

@Override

public int getOpacity() {

return PixelFormat.TRANSLUCENT;

}

@Override

protected void onBoundsChange(Rect bounds) {

super.onBoundsChange(bounds);

int width = bounds.width();

int height = bounds.height();

mPath.reset();

mPath.moveTo(0, height / 2);

mPath.lineTo(width / 4, 0);

mPath.lineTo(width \* 3 / 4, 0);

mPath.lineTo(width, height / 2);

mPath.lineTo(width \* 3 / 4, height);

mPath.lineTo(width / 4, height);

mPath.close();

}

}

В методе **draw()** выводим контур **mPath**, который будет сформирован в другом методе, на холст.

В методе **getOpacity()** возвращаем **TRANSLUCENT**, т.к. у нас будет непрозрачный шестиугольник, а оставшееся пространство будет прозрачным.

В методах **setAlpha()** и **setColorFilter()** настроим кисть. Эти методы можно не реализовывать и оставить пустыми, объект будет работать и без них. Но если вы захотите сделать его полупрозрачным или применить фильтр, то методы следует реализовать.

Кроме четырёх обязательных методов пришлось реализовать метод **onBoundsChange()**, который вызывается при изменении размеров. Так как нам нужно нарисовать шестиугольник на всю область, мы должны знать его размер. Здесь мы получаем ширину и высоту **Drawable** и используем их для создания фигуры шестиугольника.

Применим созданный объект к **ImageView**:

ImageView imageView = (ImageView) findViewById(R.id.imageView);

imageView.setBackground(new HexagonDrawable());

Класс BitmapDrawable

Можно сконвертировать **Bitmap** в **Drawable** с помощью **BitmapDrawable**.

Drawable drawable = new BitmapDrawable(getResources(), tempBitmap);

Допустим, мы загружаем картинку (Bitmap) из Галереи и выводим его в **ImageView**

Возьмём [готовый пример](http://android-er.blogspot.ru/2013/09/convert-bitmap-to-drawable-with.html). Подготовим разметку.

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:paddingBottom="@dimen/activity\_vertical\_margin"

android:paddingLeft="@dimen/activity\_horizontal\_margin"

android:paddingRight="@dimen/activity\_horizontal\_margin"

android:paddingTop="@dimen/activity\_vertical\_margin"

tools:context=".MainActivity" >

<Button

android:id="@+id/loadimage"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Load Image 1" />

<TextView

android:id="@+id/sourceuri"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content" />

<ImageView

android:id="@+id/result"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:adjustViewBounds="true"

android:background="@android:color/background\_dark"

android:scaleType="centerInside" />

</LinearLayout>

И код.

package ru.alexanderklimov.test;

import java.io.FileNotFoundException;

import android.net.Uri;

import android.os.Bundle;

import android.app.Activity;

import android.content.Intent;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.graphics.Canvas;

import android.graphics.drawable.BitmapDrawable;

import android.graphics.drawable.Drawable;

import android.view.View;

import android.view.View.OnClickListener;

import android.widget.Button;

import android.widget.ImageView;

import android.widget.TextView;

public class MainActivity extends Activity {

Button btnLoadImage;

TextView textSource;

ImageView imageResult;

final int RQS\_IMAGE1 = 1;

Uri source;

Bitmap bitmapMaster;

Canvas canvasMaster;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

btnLoadImage = (Button) findViewById(R.id.loadimage);

textSource = (TextView) findViewById(R.id.sourceuri);

imageResult = (ImageView) findViewById(R.id.result);

btnLoadImage.setOnClickListener(new OnClickListener() {

@Override

public void onClick(View arg0) {

Intent intent = new Intent(

Intent.ACTION\_PICK,

android.provider.MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI);

startActivityForResult(intent, RQS\_IMAGE1);

}

});

}

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

super.onActivityResult(requestCode, resultCode, data);

Bitmap tempBitmap;

if (resultCode == RESULT\_OK) {

switch (requestCode) {

case RQS\_IMAGE1:

source = data.getData();

try {

tempBitmap = BitmapFactory

.decodeStream(getContentResolver().openInputStream(

source));

// Convert bitmap to drawable

Drawable drawable = new BitmapDrawable(getResources(),

tempBitmap);

imageResult.setImageDrawable(drawable);

textSource.setText(drawable.getClass().toString());

} catch (FileNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

break;

}

}

}

}

Класс ColorDrawable

Класс **ColorDrawable** позволят указывать свойство изображения, основанное на единственном сплошном цвете. В качестве альтернативы можно использовать тег **<color>** в каталоге **res/drawable**:

<color xmlns:android="http://schemas.android.com/apk/res/android"

android:color="#FF0000" />

Класс GradientDrawable

Класс **GradientDrawable** позволяет закрашивать градиентом кнопки, фон и так далее. Каждый градиент описывает плавный переход между двумя или тремя цветами с помощью линейного или радиального алгоритма, а также с помощью метода развёртки. В ресурсах **GradientDrawable** описывается в виде тега [<gradient>](http://developer.alexanderklimov.ru/android/theory/shape.php#gradient), находясь внутри определения ресурса ShapeDrawable (**<shape>**).

[Пример](http://developer.alexanderklimov.ru/android/catshop/gradient.php#gradientdrawable)

Класс TransitionDrawable

Одна из возможностей использования объектов Drawable — загрузка картинок с плавными переходами. Создание переходов между изображениями с помощью класса TransitionDrawable лучше рассмотреть на примере. Добавим на форму элемент ImageView, в которую будут загружаться картинки:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:gravity="center">

<ImageView

android:id="@+id/image"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:src="@drawable/photo1"/>

</LinearLayout>

В каталоге res/drawable/ проекта поместим два графических файла photol.jpg и photo2.jpg. В том же каталоге res/drawable/ создадим файл transition.xml, в котором определим переход.

<?xml version="1.0" encoding="utf-8"?>

<transition xmlns:android="http://schemas.android.com/apk/res/android">

<item android:drawable="@drawable/photo1"></item>

<item android:drawable="@drawable/photo2"></item>

</transition>

В классе активности создадим объект TransitionDrawable и установим его как содержание ImageView:

Resources res = this.getResources();

mTransition = (TransitionDrawable)res.getDrawable(R.drawable.transition);

Также создадим обработчик onCiick(), при вызове которого будет происходить смена картинок с плавным переходом в течение 1 секунды:

mTransition.startTransition(1000);

Полный код примера:

public class TransitionActivity extends Activity

implements OnClickListener {

private ImageView image;

private TransitionDrawable mTransition;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

image = (ImageView)findViewById(R.id.image);

image.setOnClickListener(this);

Resources res = this.getResources();

mTransition = (TransitionDrawable)res.getDrawable(R.drawable.transition);

}

@Override

public void onClick(View v) {

image.setImageDrawable(mTransition);

mTransition.startTransition(1000);

}

}

Запустите проект. При касании экрана будет происходить плавное переключение от первого изображения к второму.

У примера есть один недостаток - при повторном касании произойдет резкая смена на первую картинку, которая снова будет плавно переключаться на вторую. Для обратной анимации от второй картинки к первой нужно задействовать метод **reverseTransition()**:

int count = 2;

@Override

public void onClick(View v) {

image.setImageDrawable(mTransition);

if(count%2 == 0){

mTransition.startTransition(1000);

count++;

}

else{

mTransition.reverseTransition(1000);

count++;

}

}

Класс ShapeDrawable

Класс **ShapeDrawable**, входящий в пакет **android.graphics.drawable**, позволяет определять различные свойства рисунка с помощью различных методов, например **setAlpha()** — для установки прозрачности, **setColorFilter()** и т.д.

С объектами **ShapeDrawable** вы можете программно создавать любые примитивные формы и их стили. Для работы через XML используется тег[<shape>](http://developer.alexanderklimov.ru/android/theory/shape.php#shape). В библиотеке Android есть несколько классов, производных от базового класса **Shape**:

* PathShape;
* RectShape;
* ArcShape;
* OvalShape;
* RoundRectShape

1. Класс RectShape

Для рисования прямоугольника используется класс **RectShape**. Также необходимо установить цвет и границы фигуры. Если вы не установите границы, то прямоугольник не будет рисоваться. Если вы не установите цвет, фигура будет черной (значение по умолчанию).

Если задать высоту или ширину в 1-2 пиксела, то можно получить не прямоугольник, а вертикальную или горизонтальную линию:

ShapeDrawable line = new ShapeDrawable(new RectShape());

line.setIntrinsicHeight(2);

line.setIntrinsicWidth(150);

line.getPaint().setColor(Color.MAGENTA);

1. Класс OvalShape

Для рисования эллипсов используется класс **OvalShape**:

ShapeDrawable oval = new ShapeDrawable(new OvalShape());

oval.setIntrinsicHeight(100);

oval.setIntrinsicWidth(150);

oval.getPaint().setColor(Color.RED);

1. Класс RoundRect

Прорисовка прямоугольника с закругленными сторонами (RoundRect) несколько сложнее. Конструктор класса **RoundRect** для рисования прямоугольника с закругленными сторонами принимает несколько параметров:

RoundRectShape(float[] outerRadii, RectF inset, float[] innerRadii)

* outerRadii — массив из восьми значений радиуса закругленных сторон внешнего прямоугольника. Первые два значения для верхнего левого угла, остальные пары отсчитываются по часовой стрелке от первой пары. Если на внешнем прямоугольнике закруглений не будет, передается null;
* inset — объект класса RectF, который определяет расстояние от внутреннего прямоугольника до каждой стороны внешнего прямоугольника. Конструктор RectF принимает четыре параметра: пары X и Y координат левого верхнего и правого нижнего углов внутреннего прямоугольника. Если внутреннего прямоугольника нет, передается null;
* innerRadii — массив из восьми значений радиуса закруглений углов для внутреннего прямоугольника. Первые два значения — координаты верхнего левого угла, остальные пары отсчитываются по часовой стрелке от первой пары. Если на внутреннем прямоугольнике закруглений не будет, передается null.

float[] outR = new float[] {6, 6, 6, 6, 6, 6, 6, 6 };

RectF rectF = newRectF(8, 8, 8, 8);

float[] inR new float[] { 6, 6, 6, 6, 6, 6, 6, 6 };

ShapeDrawable shape = new ShapeDrawable(

new RoundRectShape(outR, rectF, inR));

shape.setIntrinsicHeight(120);

shape.setIntrinsicWidth(150);

shape.getPaint().setColor(Color.BLUE);

1. Класс Path

Класс **Path** формирует множественный контур геометрических путей, состоящих из прямых линейных сегментов, квадратичных и кубических кривых. Для установки точек и перемещения линий (или кривых) используются методы moveTo() и lineTo(). Например, с помощью класса Path можно нарисовать пятиконечную звезду:

Path p = new Path();

p.moveTo(50, 0);

p.lineTo(25, 100);

p.lineTo(100,50);

p.lineTo(0,50);

p.lineTo(75,100);

p.lineTo(50,0);

ShapeDrawable d = new ShapeDrawable(new PathShape(p, 100, 100));

d.setIntrinsicHeight(100);

d.setIntrinsicWidth(100);

d.getPaint().setColor(Color.YELLOW);

d.getPaint().setStyle(Paint.Style.STROKE);

1. Класс ArcShape

Класс **ArcShape** создает фигуру в форме дуги. Принимает два параметра:угол начала прорисовки дуги в градусах и угловой размер дуги в градусах

ArcShape(float startAngle, float sweepAngle)

ScaleDrawable

Класс **ScaleDrawable** используется для масштабирования.

RotateDrawable

Класс **RotateDrawable** используется для поворотов.

LayerDrawable

**LayerDrawable** позволяет накладывать объекты **Drawable** друг на друга. Создав массив из полупрозрачных картинок, вы можете наложить их один поверх другого. Подготовьте три полупрозрачных изображения круга (red.png, blue.png, green.png) и поместите их в ресурсы **res/drawable**. Создадим фон для макета при помощи наложения этих элементов.

// В onCreate()

LinearLayout linear = (LinearLayout)findViewById(R.id.linear);

linear.setBackground(createLayerDrawable());

private LayerDrawable createLayerDrawable() {

BitmapDrawable drawable0 = (BitmapDrawable) getResources().getDrawable(

R.drawable.blue);

drawable0.setGravity(Gravity.CENTER\_VERTICAL);

BitmapDrawable drawable1 = (BitmapDrawable) getResources().getDrawable(

R.drawable.red);

drawable1.setGravity(Gravity.LEFT);

BitmapDrawable drawable2 = (BitmapDrawable) getResources().getDrawable(

R.drawable.green);

drawable2.setGravity(Gravity.RIGHT);

Drawable drawableArray[] = new Drawable[] { drawable0, drawable1, drawable2 };

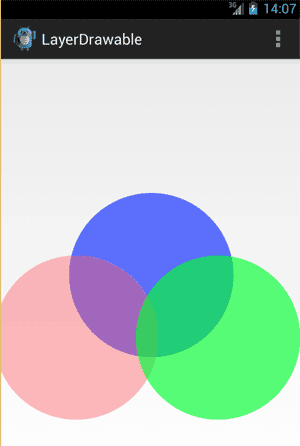
LayerDrawable layerDraw = new LayerDrawable(drawableArray);

layerDraw.setLayerInset(1, -20, 300, 0, 0); // смещаем второй слой

layerDraw.setLayerInset(2, 100, 300, 0, 0); // смещаем третий слой

return layerDraw;

}



Также можно создать объекты **LayerDrawable** через XML с помощью тега [layer-list](http://developer.alexanderklimov.ru/android/theory/shape.php#layer-list).

StateListDrawable

У некоторых элементов управления есть некоторые состояния: нажато, выбрано и т.д.. В Android есть возможность задавать изображения для любого из таких состояний при помощи класса **StateListDrawable**. Но удобнее это делать через [XML-ресурсы](http://developer.alexanderklimov.ru/android/theory/shape.php#state-list).

LevelListDrawable

Используя **LevelListDrawable**, вы можете эффективно размещать ресурсы **Drawable** один поверх другого, указывая целочисленный индекс для каждого слоя. [Пример с использованием XML](http://developer.alexanderklimov.ru/android/theory/shape.php#levellistdrawable).

Дополнительное чтение

[Звёзды из котов - создаём собственный Drawable](http://developer.alexanderklimov.ru/android/catshop/starbitmapdrawable.php)

[Слайды про Drawable](https://speakerdeck.com/cyrilmottier/mastering-android-drawables) - там есть коты, советы по использованию **Drawable** в собственных **View**, совет по размещению логотипа в качестве фона активности через тему приложения.