**[Урок 162. Графика. Drawable. Shape, Gradient.](http://startandroid.ru/ru/uroki/vse-uroki-spiskom/377-urok-162-grafika-drawable-shape-gradient.html)**

В этом уроке:

- изучаем drawable-тег shape

Подробно ознакомившись с Bitmap, переходим к другому ключевому объекту графики – Drawable.

Drawable – это абстрактный контейнер для графического объекта. Его главное абстрактное умение – он может нарисовать свое содержимое на предоставленной ему канве. А вот что конкретно он нарисует, зависит уже от реализации. Это может быть, например, тот же Bitmap, если мы используем BitmapDrawable объект. Или это может быть простая геометрическая фигура, если мы используем ShapeDrawable.

Drawable-объекты мы можем создавать сами напрямую в коде. Но для некоторых из них мы можем создать описание в xml-файлах, в папке res/drawable. И когда он нам понадобится, мы укажем id файла, система сама распарсит его и создаст нам нужный объект.

Самое распространенное использование Drawable – это свойство background, которое есть у каждого View. В качестве значения вы можете указать там RGB-цвет или id ресурса из папки res/drawable. Далее система сама по этому значению определит тип и далее:

- если это цвет, то создаст ColorDrawable,   
- если это id картинки в res/drawable, то создаст BitmapDrawable  
- если это id xml-файла в res/drawable, то распарсит его и вернет соответствующего ему наследника Drawable: StateListDrawable, AnimationDrawable или какой-то другой.

В итоге View получит свой Drawable-объект и сможет его нарисовать.

В общем, как вы поняли, у абстрактного Drawable есть несколько наследников-реализаций и в ближайших уроках мы их рассмотрим. Начнем с тех, которые можно описать в xml. По ним есть отдельная статья в [хелпе](http://developer.android.com/guide/topics/resources/drawable-resource.html" \t "_blank). Там, правда, есть пара ошибок копипаста, но в остальном все верно.

В этом уроке рассмотрим тег <**shape**>

Создадим проект:

**Project name**: P1621\_DrawableShape   
**Build Target**: Android 4.4   
**Application name**: DrawableShape  
**Package name**: ru.startandroid.develop.p1621drawableshape  
**Create Activity**: MainActivity

res/layout/**main.xml**:

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<RelativeLayout  
 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:padding=*"16dp"*>  
<ImageView  
 android:id=*"@+id/imageView"*  
 android:layout\_width=*"200dp"*  
 android:layout\_height=*"200dp"*  
 android:layout\_centerInParent=*"true"*  
 android:background=*"#cccccc"*  
 android:scaleType=*"center"*>  
</ImageView>  
</RelativeLayout>

Создадим файл

res/drawable/**shape.xml**:

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"rectangle"*>  
<stroke  
 android:width=*"1dp"*  
 android:color=*"#000000"*>  
</stroke>  
</shape>

Корневой тег shape и у него же есть одноименный атрибут shape, в котором мы указываем тип фигуры. Мы указали **rectangle** – это прямоугольник.

Далее, внутри тега shape, идет тег stroke, который позволяет задать нам характеристики линии контура (периметра) фигуры. Мы задаем толщину (width) в 1dp и черный цвет (color).

**MainActivity.java**:

**package** ru.startandroid.develop.p1621drawableshape;  
  
**import** android.app.Activity;  
**import** android.os.Bundle;  
**import** android.widget.ImageView;  
  
**public class** MainActivity **extends** Activity {  
  
ImageView imageView;  
    
  @Override  
  **protected void** onCreate(Bundle savedInstanceState) {  
    **super**.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
    imageView = (ImageView) findViewById(R.id.imageView);  
    setDrawable();  
  }  
    
  **private void** setDrawable() {  
  }  
  
}

В ImageView пока ничего не отображаем.

Запускаем приложение



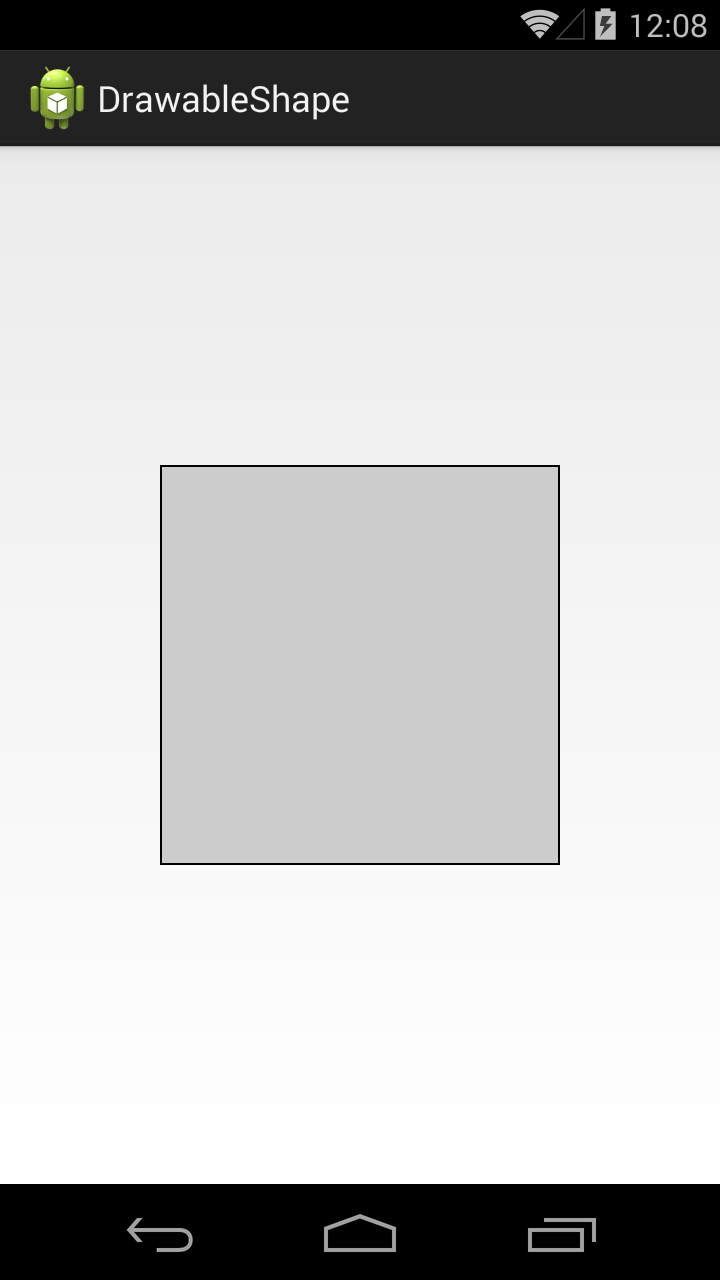
Видим ImageView с серым фоном

Перепишем метод setDrawable:

**private void** setDrawable() {  
    imageView.setImageResource(R.drawable.shape);  
  }

В качестве drawable будем передавать наш файл shape

Результат:



ImageView теперь отображает прямоугольник с черным контуром.

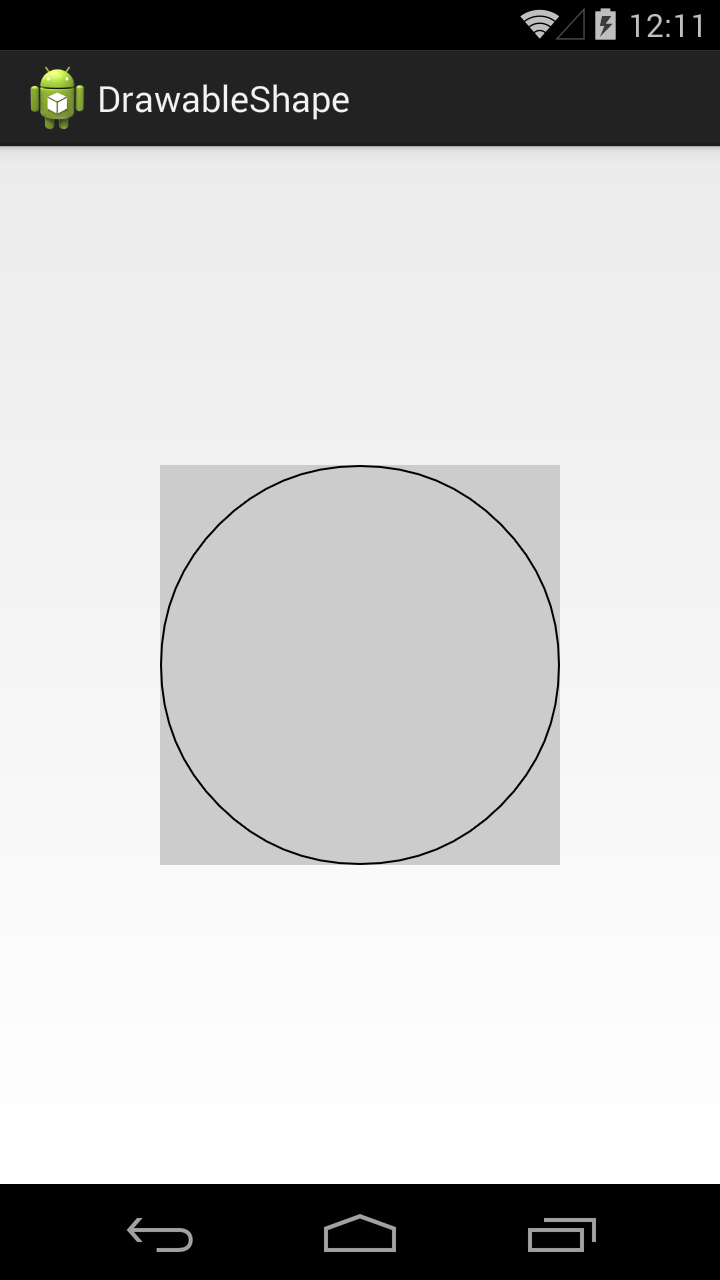
Посмотрим, какие еще фигуры нам доступны.

Перепишем shape.xml:

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"oval"*>  
<stroke  
 android:width=*"1dp"*  
 android:color=*"#000000"*>  
</stroke>  
</shape>

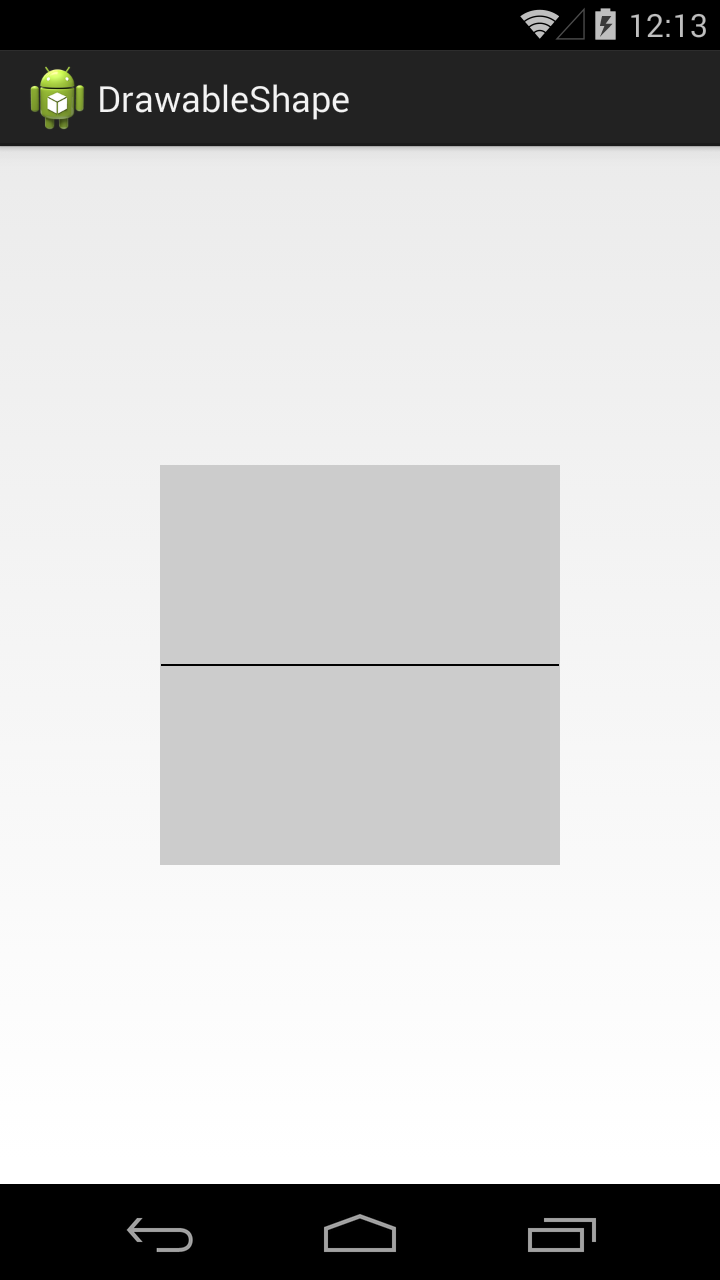
Значение атрибута shape = **oval**, это эллипс

В нашем случае получился круг, т.к. ImageView квадратный.



Значение **line** даст нам горизонтальную линию

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"line"*>  
<stroke  
 android:width=*"1dp"*  
 android:color=*"#000000"*>  
</stroke>  
</shape>

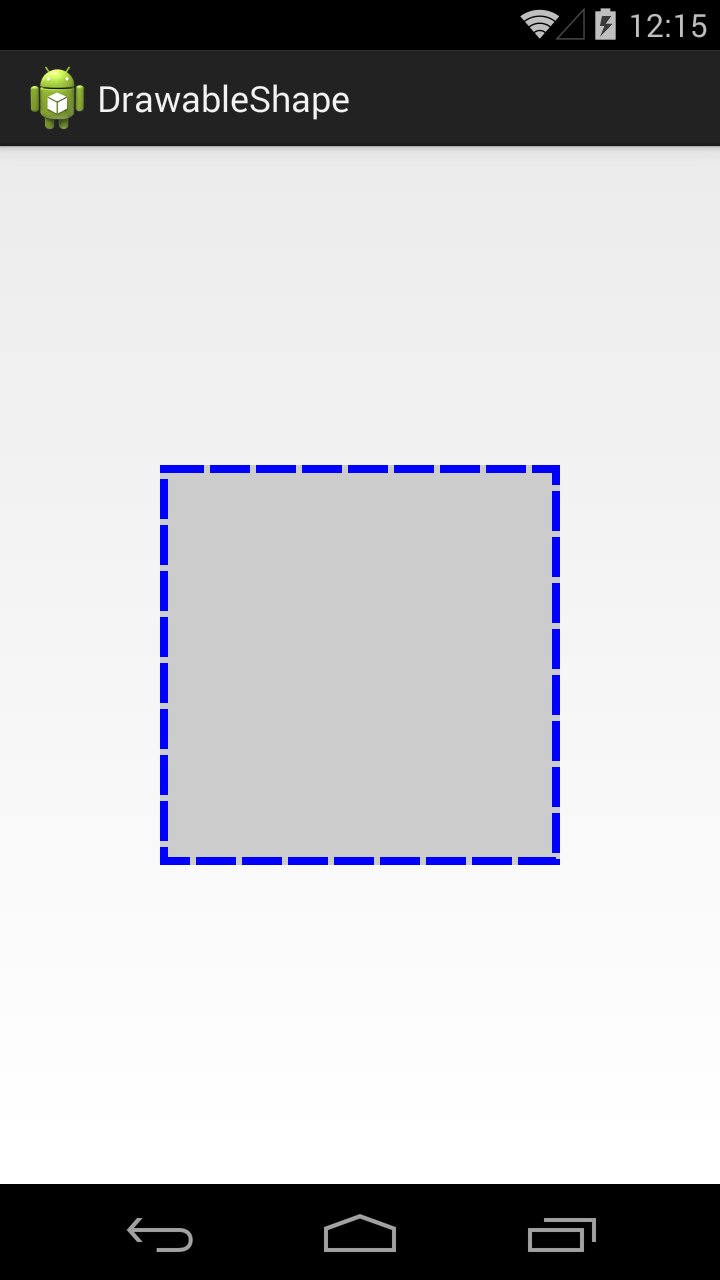


Есть еще фигура кольцо (**ring**), но о нем чуть позже.

Вернемся к тегу stroke. Добавим в него параметров.

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"rectangle"*>  
<stroke  
 android:width=*"4dp"*  
 android:color=*"#0000FF"*  
 android:dashWidth=*"20dp"*  
 android:dashGap=*"3dp"*>  
</stroke>  
</shape>

Ширина – 4dp, цвет – синий. Параметры dashWidth и dashGap сделают линию контура пунктирной. dashWidth задает длину пунктирной черточки, а dashGap – расстояние между черточками

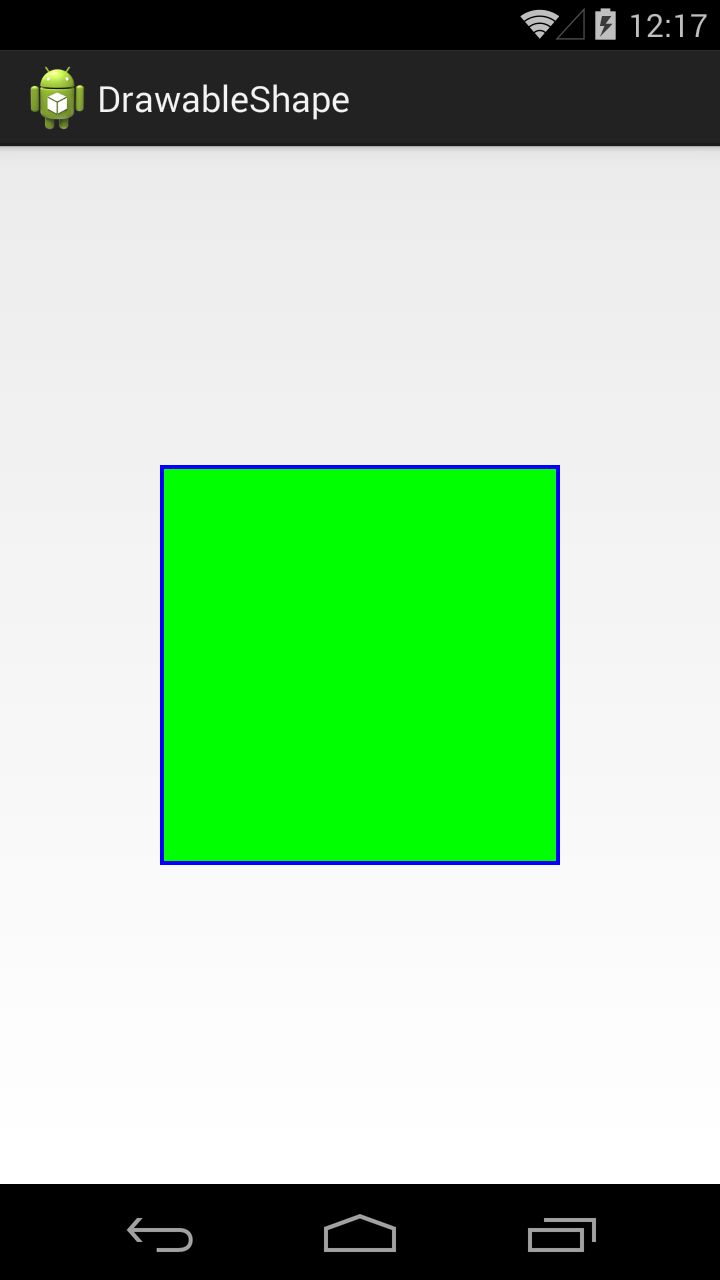


### <solid>

Добавим заливку, для этого используется тег solid

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"rectangle"*>  
<stroke  
 android:width=*"2dp"*  
 android:color=*"#0000FF"*>  
</stroke>  
<solid  
 android:color=*"#00FF00"*>  
</solid>  
</shape>

Тег solid имеет атрибут color, который позволяет указать цвет заливки фигуры. Мы указали в нем зеленый цвет.

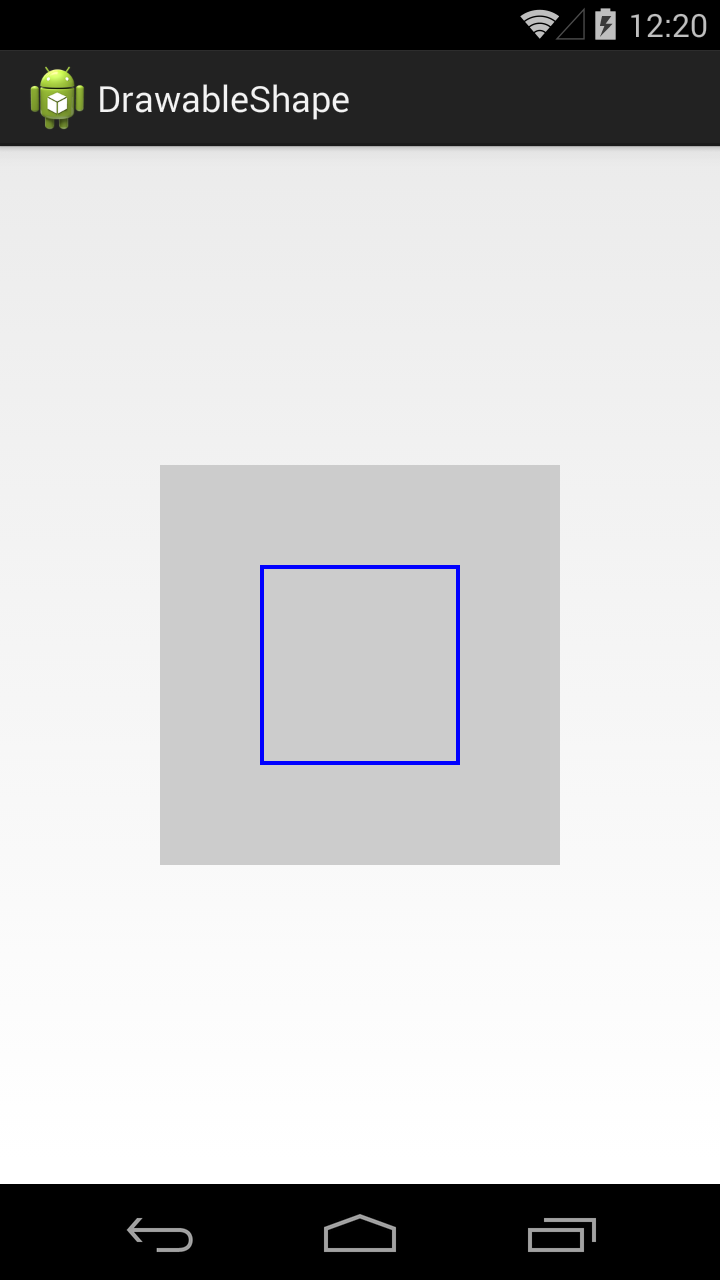


**<size>**

По умолчанию фигура занимает все доступное ей пространство, но мы можем явно указать ее размер с помощью тега size.

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"rectangle"*>  
<stroke  
 android:width=*"2dp"*  
 android:color=*"#0000FF"*>  
</stroke>  
<size  
 android:width=*"100dp"*  
 android:height=*"100dp"*>  
</size>  
</shape>

Используем тег size с атрибутами width и height



Фигура теперь размером 100х100 и уже не занимает всю доступную ей площадь.

Учтите, что режим отображения зависит от scaleType y ImageView.

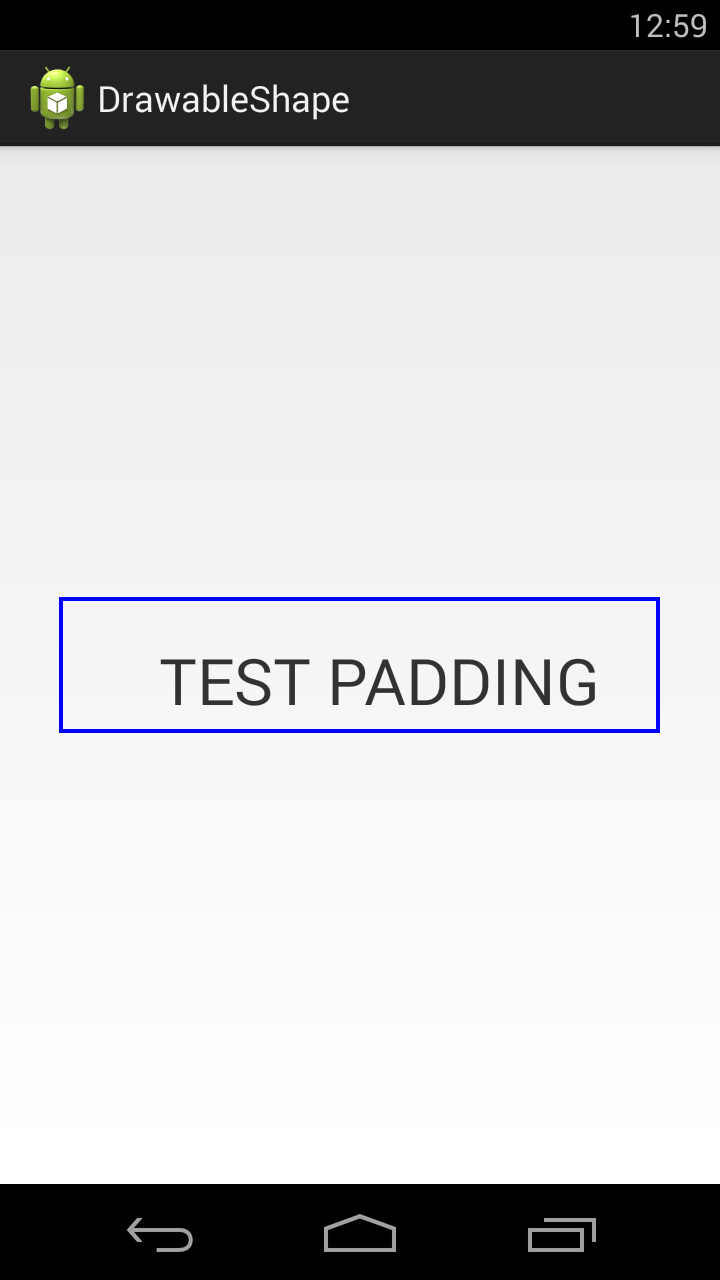
### <padding>

Тег padding позволяет нам задать величину отступа внутри фигуры. Это актуально, например, для TextView. Отступ будет учтен при размещении текста.

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"rectangle"*>  
<stroke  
 android:width=*"2dp"*  
 android:color=*"#0000FF"*>  
</stroke>  
<padding  
 android:bottom=*"5dp"*  
 android:top=*"20dp"*  
 android:left=*"50dp"*  
 android:right=*"30dp"*>  
</padding>  
</shape>

Мы указали различные отступы со всех 4 сторон.

Если мы теперь повесим эту фигуру в качестве background для TextView, результат будет таким



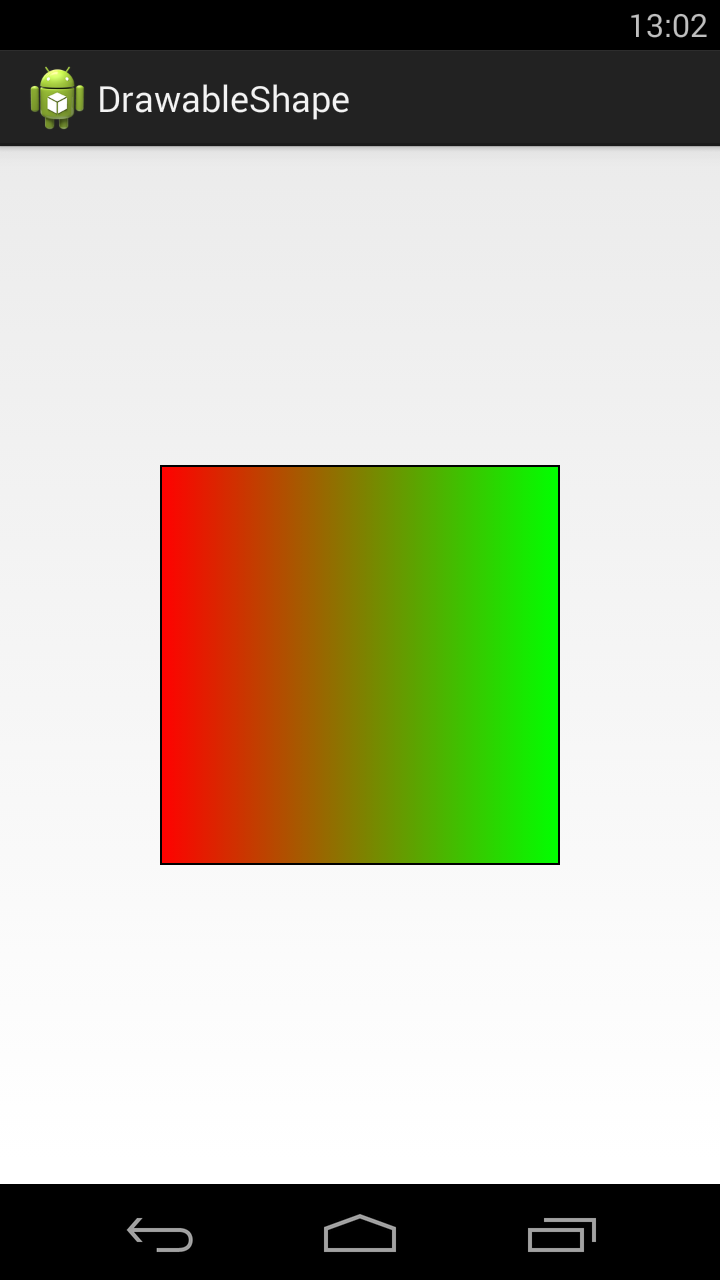
### <gradient>

В качестве заливки мы можем использовать не один цвет, а градиент из двух или трех. Для этого используется тег gradient.

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"rectangle"*>  
<stroke  
 android:width=*"1dp"*  
 android:color=*"#000000"*>  
</stroke>  
<gradient  
 android:startColor=*"#FF0000"*  
 android:endColor=*"#00FF00"*>  
</gradient>  
</shape>

В теге gradient указываем два атрибута-цвета: startColor и endColor.

В результате получится градиент, переходящий из первого цвета во второй.

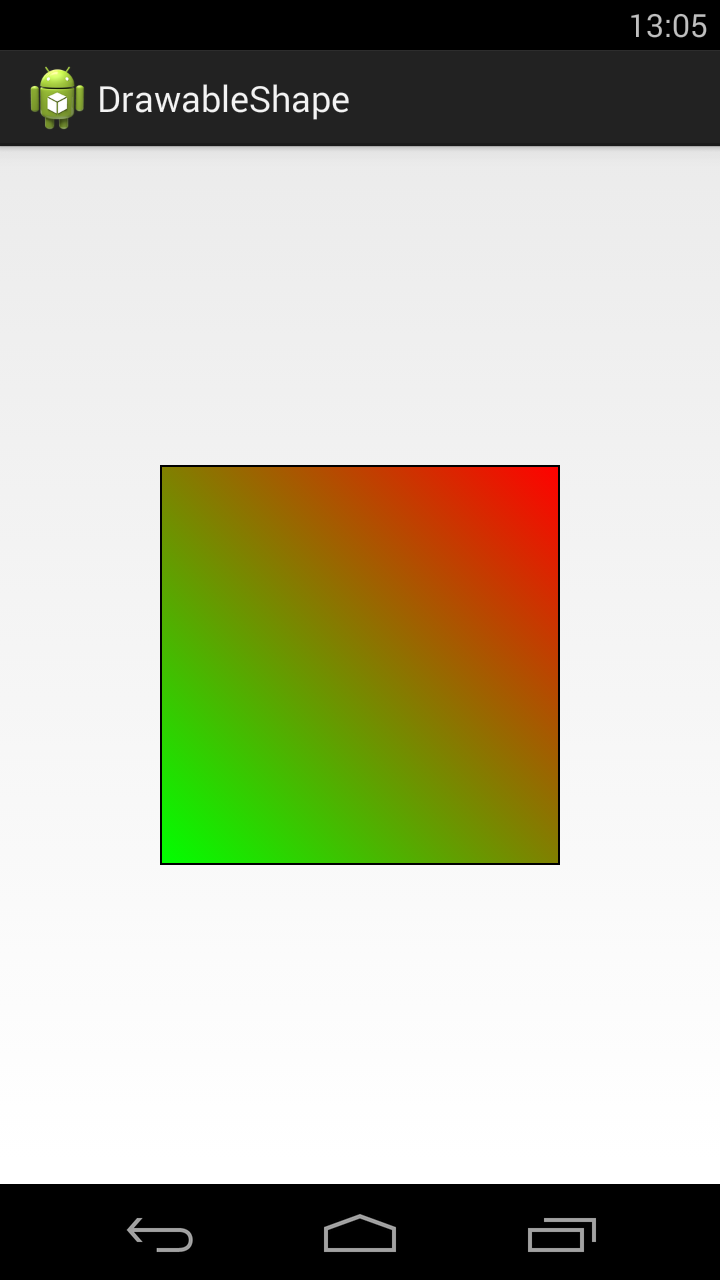


Градиент вовсе необязательно должен идти слева-направо. Мы можем указать угол направления. Для этого у тега gradient есть атрибут angle

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"rectangle"*>  
<stroke  
 android:width=*"1dp"*  
 android:color=*"#000000"*>  
</stroke>  
<gradient  
 android:startColor=*"#FF0000"*  
 android:endColor=*"#00FF00"*  
 android:angle=*"225"*>  
</gradient>  
</shape>

В angle указываем угол 225. Угол 0 означает направление слева-направо, 90 – снизу вверх и т.д. Угол должен быть кратным 45.

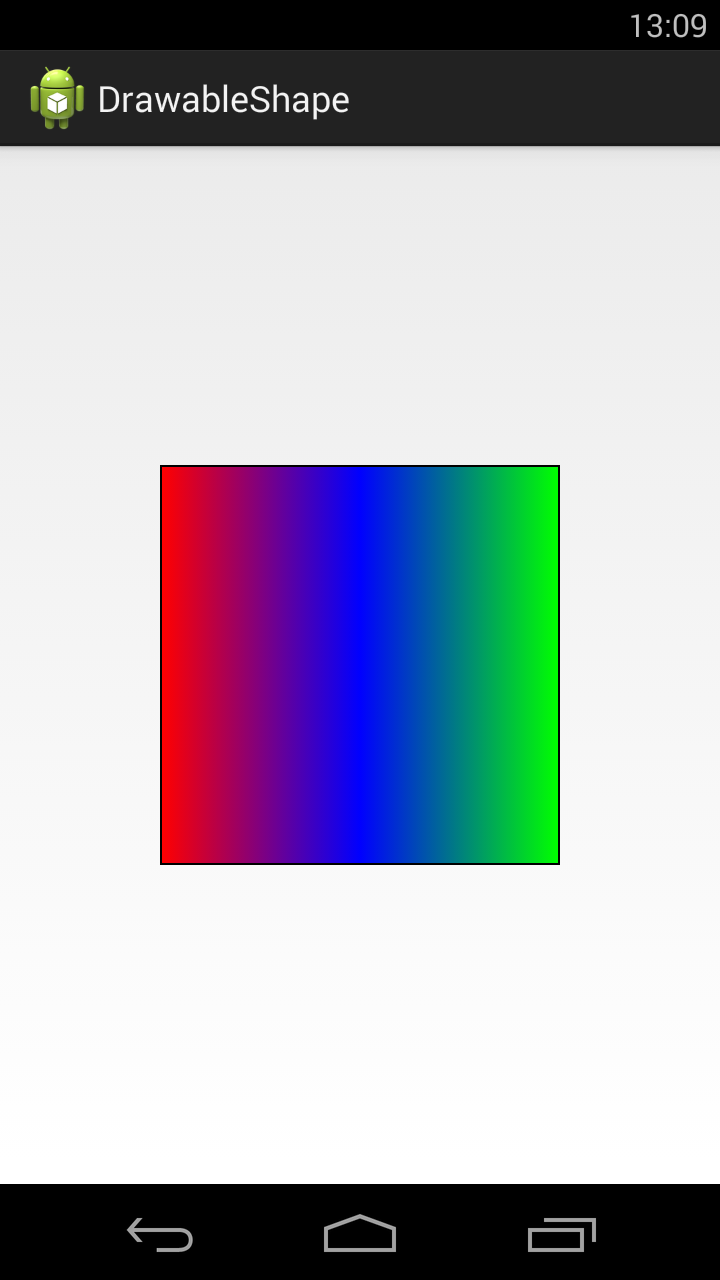
В результате видим угол справа-сверху налево-вниз.



Тег gradient позволяет указать третий цвет, который вклинится между start- и end- цветами.

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"rectangle"*>  
<stroke  
 android:width=*"1dp"*  
 android:color=*"#000000"*>  
</stroke>  
<gradient  
 android:startColor=*"#FF0000"*  
 android:endColor=*"#00FF00"*  
 android:centerColor=*"#0000FF"*>  
</gradient>  
</shape>

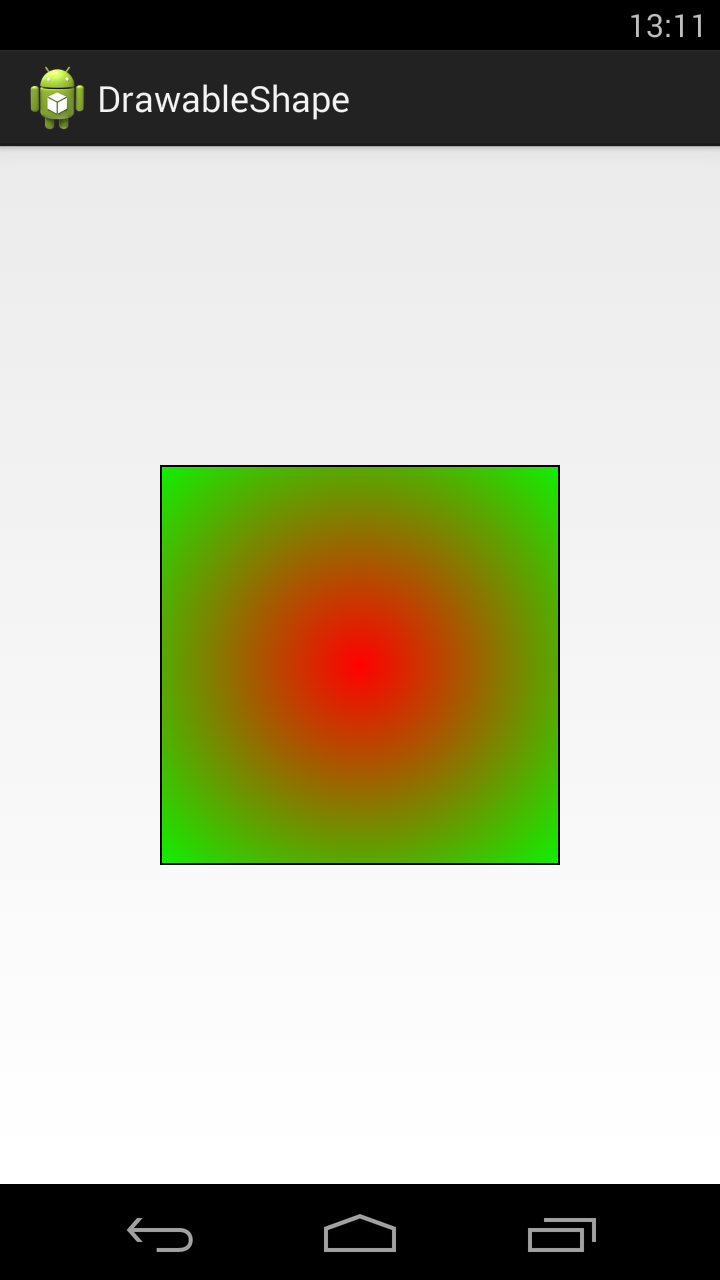
В атрибуте centerColor укажем синий цвет, и он в градиенте будет между красным и зеленым.



Градиент может быть разных типов. Мы рассмотрели тип linear, который используется по умолчанию. Есть еще два типа: radial и sweep.

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"rectangle"*>  
<stroke  
 android:width=*"1dp"*  
 android:color=*"#000000"*>  
</stroke>  
<gradient  
 android:type=*"radial"*  
 android:gradientRadius=*"300"*  
 android:startColor=*"#FF0000"*  
 android:endColor=*"#00FF00"*>  
</gradient>  
</shape>

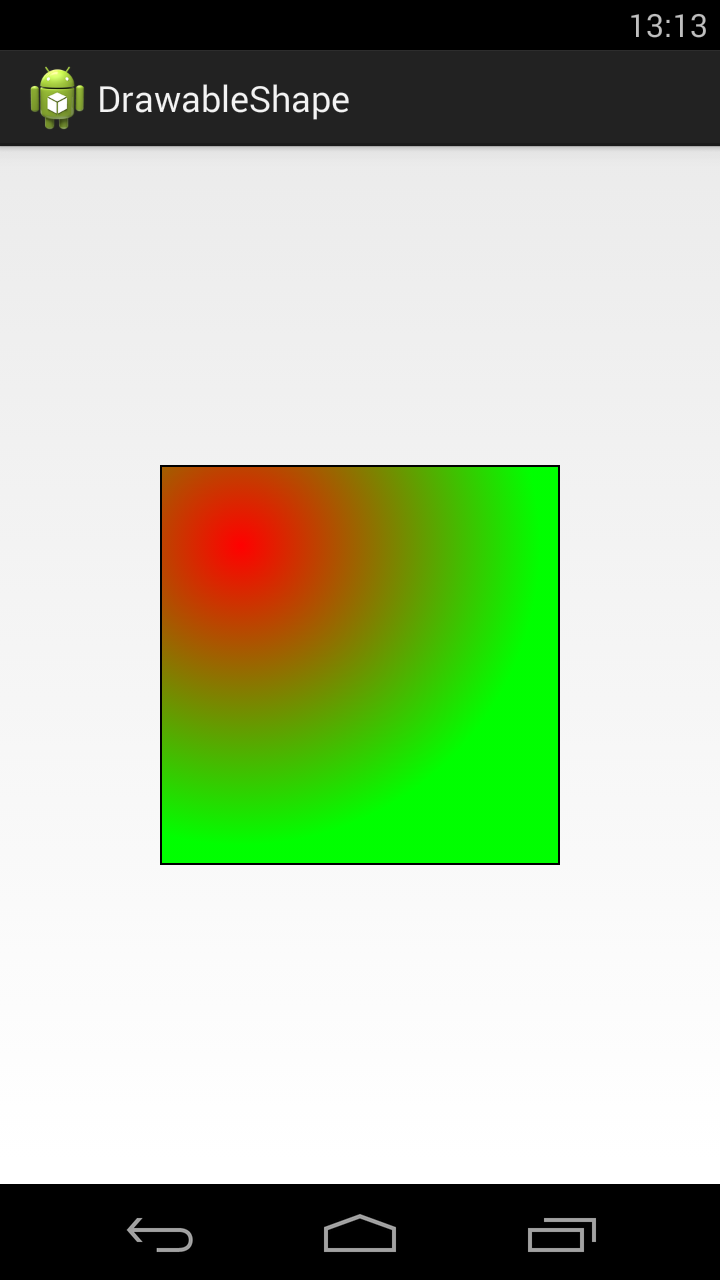
Тип radial даст нам круговой градиент, а в параметре gradientRadius мы должны указать радиус круга.



Мы можем указать точку центра кругового градиента атрибутами centerX и centerY. Значения этих атрибутов должны быть от 0 до 1.

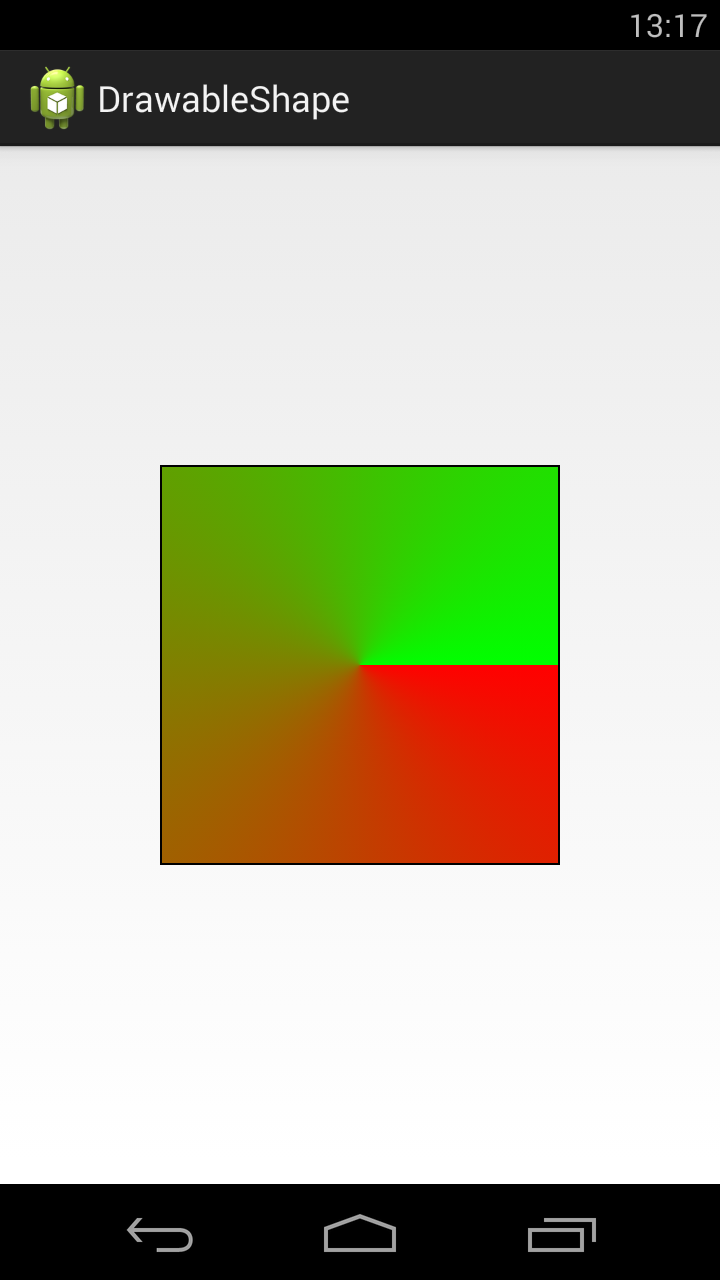
<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"rectangle"*>  
<stroke  
 android:width=*"1dp"*  
 android:color=*"#000000"*>  
</stroke>  
<gradient  
 android:type=*"radial"*  
 android:gradientRadius=*"300"*  
 android:startColor=*"#FF0000"*  
 android:endColor=*"#00FF00"*  
 android:centerX=*"0.2"*  
 android:centerY=*"0.2"*>  
</gradient>  
</shape>

Центра градиента будет в точке (0.2, 0.2), если принять размеры фигуры за единицу.



Теперь посмотрим, как выглядит градиент sweep.

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"rectangle"*>  
<stroke  
 android:width=*"1dp"*  
 android:color=*"#000000"*>  
</stroke>  
<gradient  
 android:type=*"sweep"*  
 android:startColor=*"#FF0000"*  
 android:endColor=*"#00FF00"*>  
</gradient>  
</shape>



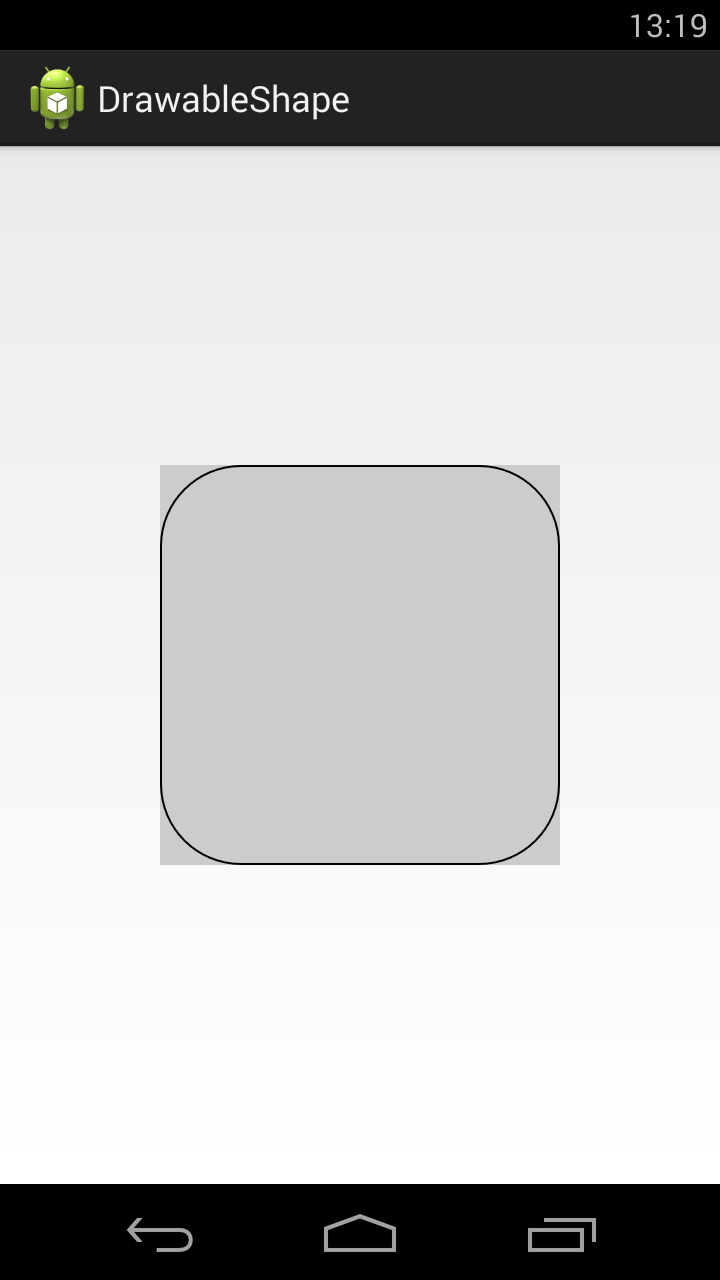
Для этого типа градиента также можно использовать атрибуты centerColor, centerX и centerY.

### <corners>

Для фигуры прямоугольника мы можем сгладить углы. За это отвечает тег corners.

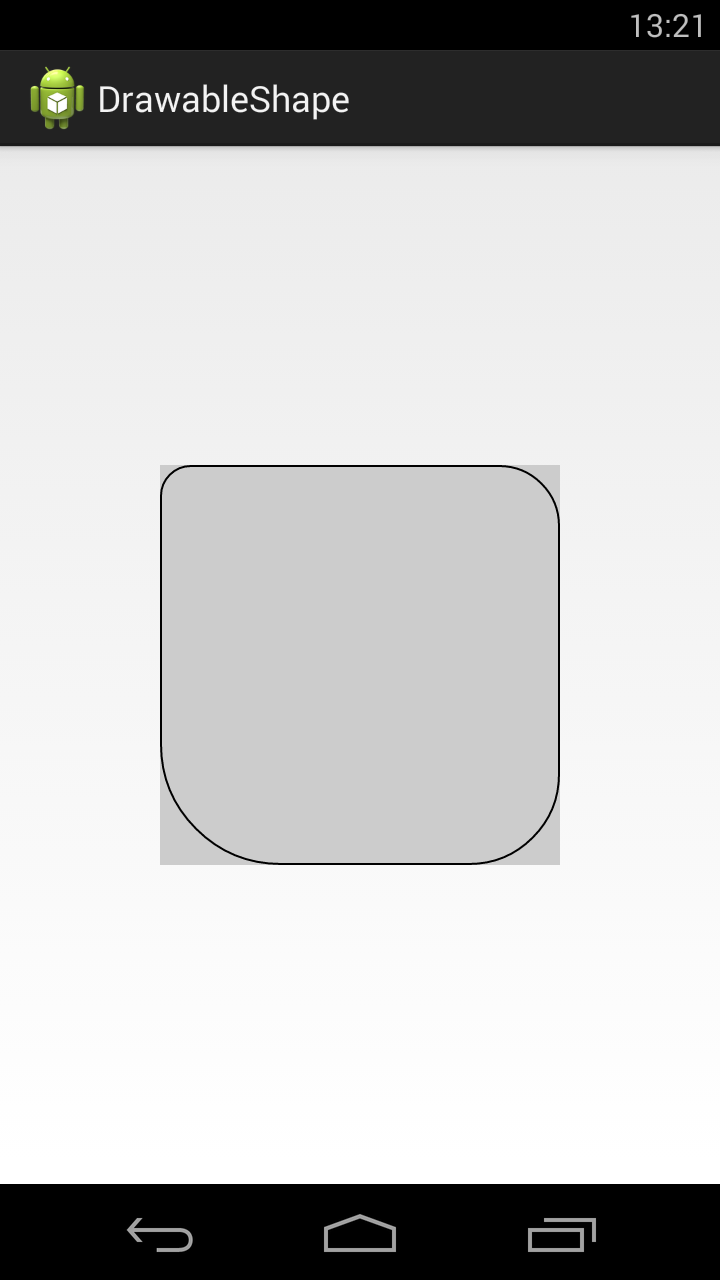
<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"rectangle"*>  
<stroke  
 android:width=*"1dp"*  
 android:color=*"#000000"*>  
</stroke>  
<corners  
 android:radius=*"40dp"*>  
</corners>  
</shape>

Атрибут radius позволяет задать радиус закругления сразу для всех углов.



Есть возможность задать свой радиус для каждого угла отдельно.

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"rectangle"*>  
<stroke  
 android:width=*"1dp"*  
 android:color=*"#000000"*>  
</stroke>  
<corners  
 android:topLeftRadius=*"15dp"*  
 android:topRightRadius=*"30dp"*  
 android:bottomRightRadius=*"45dp"*  
 android:bottomLeftRadius=*"60dp"*>  
</corners>  
</shape>



### Кольцо

Нам осталось рассмотреть четвертую фигуру - кольцо. Чтобы его получить, надо в атрибуте shape указать значение ring. Для кольца мы можем настроить два параметра: размер внутреннего радиуса и толщина кольца. Причем, эти два параметра мы можем указывать в абсолютном и относительном выражении.

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:innerRadius=*"50dp"*  
 android:shape=*"ring"*  
 android:thickness=*"40dp"*  
 android:useLevel=*"false"*>  
<solid  
 android:color=*"#FF0000"*>  
</solid>  
<size  
 android:height=*"200dp"*  
 android:width=*"200dp"*>  
</size>  
</shape>

innerRadius – позволяет указать внутренний радиус, а thickness – толщину кольца. Атрибут useLevel, который нам пока неизвестен, должен быть false, иначе эта фигура у меня не отображалась.



Отобразился круг с внутренним радиусом = 50dp и толщиной = 40dp.

Попробуем указать толщину кольца в относительном значении. Для этого вместо thickness используем thicknessRatio. В этом атрибуте мы указываем во сколько раз толщина кольца будет меньше его ширины.

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:innerRadius=*"50dp"*  
 android:shape=*"ring"*  
 android:thicknessRatio=*"10"*  
 android:useLevel=*"false"*>  
<solid  
 android:color=*"#FF0000"*>  
</solid>  
<size  
 android:height=*"200dp"*  
 android:width=*"200dp"*>  
</size>  
</shape>

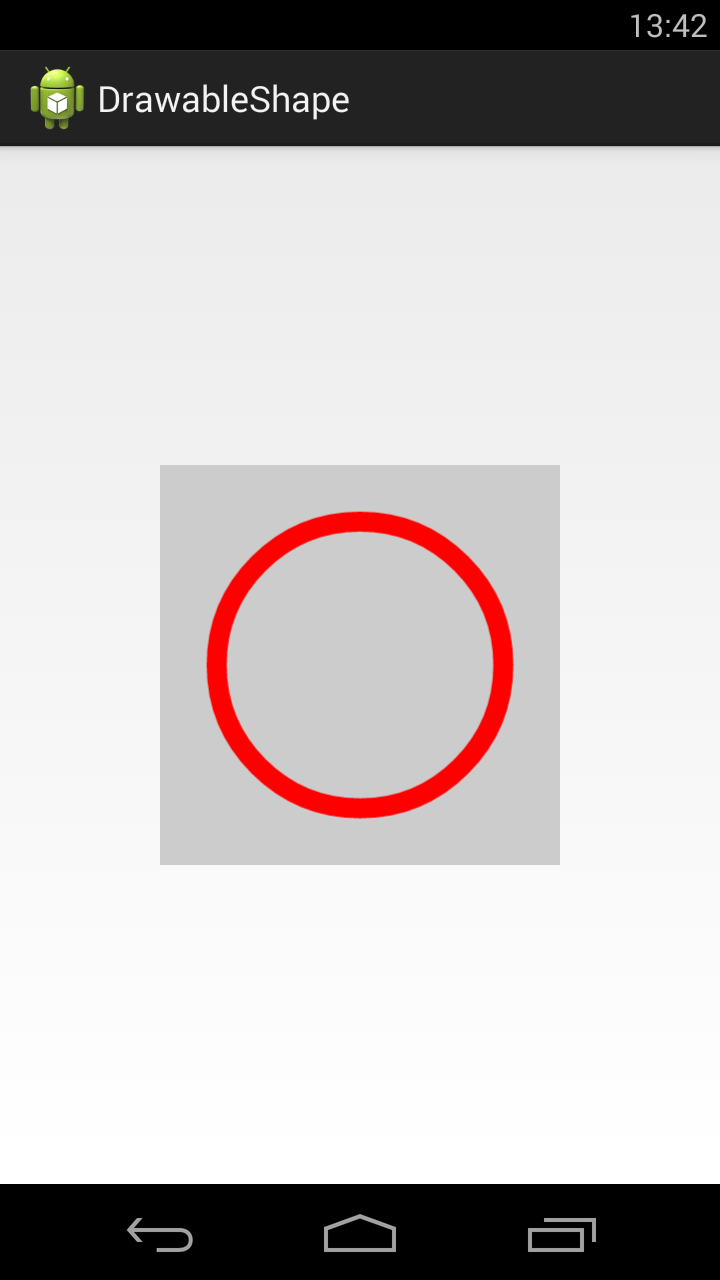
Ширина кольца = 200 dp, это указано в теге size. thicknessRatio =10, значит толщина кольца = 200 dp / 10 = 20dp.



Теперь укажем внутренний радиус в относительном выражении. Для этого вместо innerRadius используем innerRadiusRatio. В атрибуте innerRadiusRatio указываем во сколько раз внутренний радиус меньше ширины кольца.

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:innerRadiusRatio=*"3"*  
 android:shape=*"ring"*  
 android:thickness=*"10dp"*  
 android:useLevel=*"false"*>  
<solid  
 android:color=*"#FF0000"*>  
</solid>  
<size  
 android:height=*"200dp"*  
 android:width=*"200dp"*>  
</size>  
</shape>

Ширина кольца = 200 dp. innerRadiusRatio = 3, значит внутренний радиус кольца = 200 dp / 3 = 67dp.

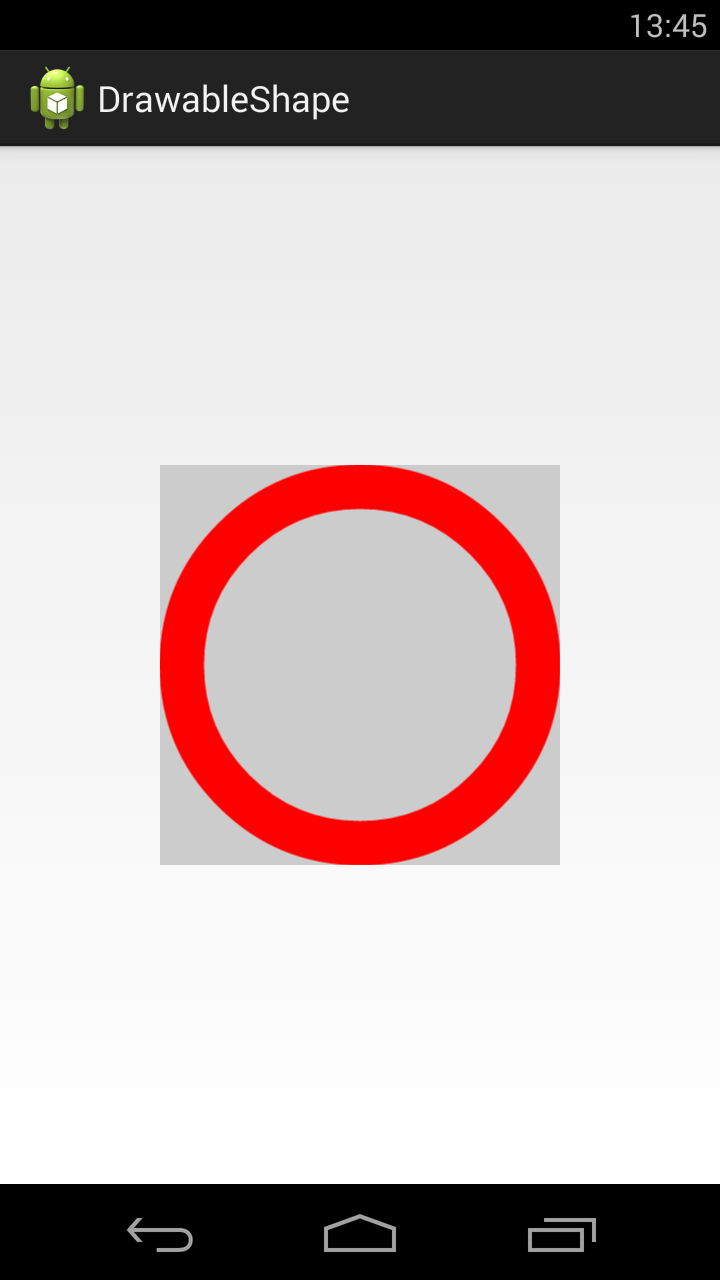


Как видите, кольцо может занимать не весь свой размер. Это зависит от значений, которые мы задаем для внутреннего радиуса и толщины.

У атрибутов относительного размера есть значения по умолчанию. Т.е. если мы явно не укажем значение для innerRadiusRatio, то по умолчанию он будет равен 3, а thicknessRatio по умолчанию равен 9. Посмотрим, как это выглядит

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:innerRadius=*"78dp"*  
 android:shape=*"ring"*  
 android:useLevel=*"false"*>  
<solid  
 android:color=*"#FF0000"*>  
</solid>  
<size  
 android:height=*"200dp"*  
 android:width=*"200dp"*>  
</size>  
</shape>

Мы указали только внутренний радиус. А размер толщины будет вычислен исходя из значения thicknessRatio по умолчанию, т.е. 9.



Теперь не будем указывать инфу о внутреннем радиусе.

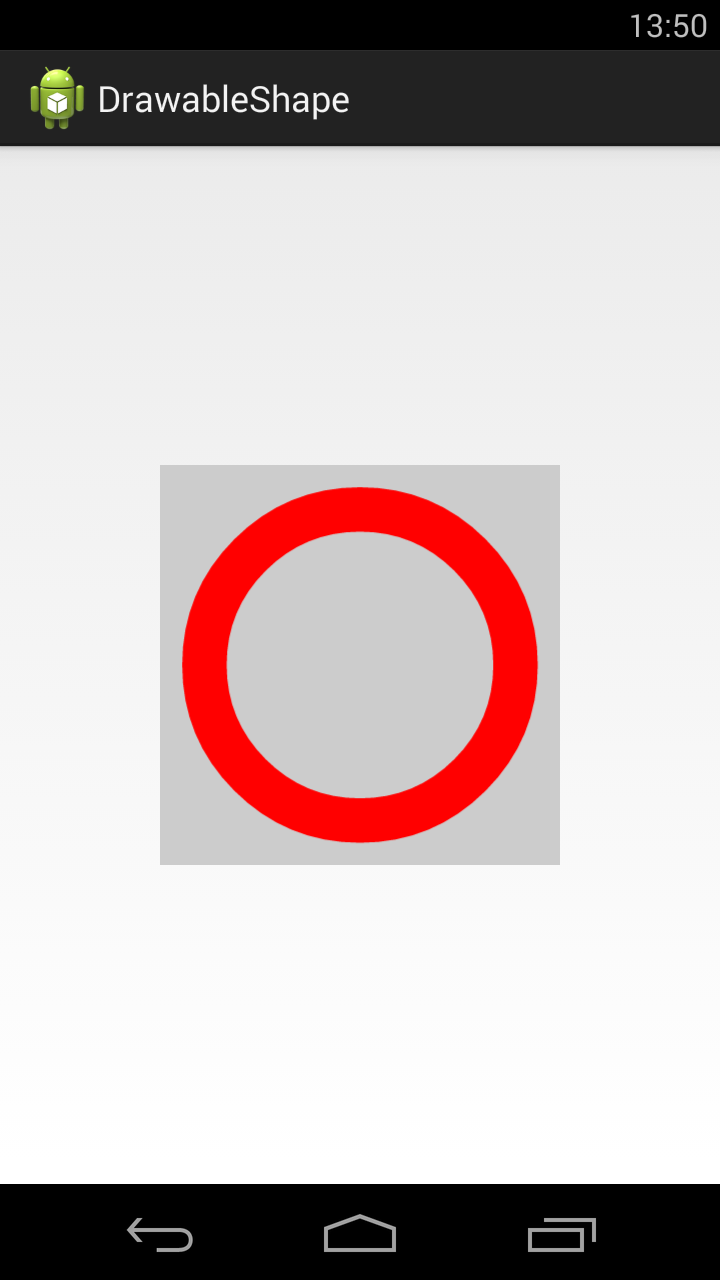
<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"ring"*  
 android:thickness=*"33dp"*  
 android:useLevel=*"false"*>  
<solid  
 android:color=*"#FF0000"*>  
</solid>  
<size  
 android:height=*"200dp"*  
 android:width=*"200dp"*>  
</size>  
</shape>

Мы указали только толщину кольца, а внутренний радиус будет вычислен исходя из значения innerRadiusRatio по умолчанию = 3.



Давайте теперь попробуем вообще не указывать размеры внутреннего радиуса и толщины, и посмотрим, что получится.

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"ring"*  
 android:useLevel=*"false"*>  
<solid  
 android:color=*"#FF0000"*>  
</solid>  
<size  
 android:height=*"200dp"*  
 android:width=*"200dp"*>  
</size>  
</shape>



Видим, что кольцо заняло не все 200 dp по высоте, которые мы ему задали в теге size. Почему? Давайте считать исходя из значения по умолчанию.

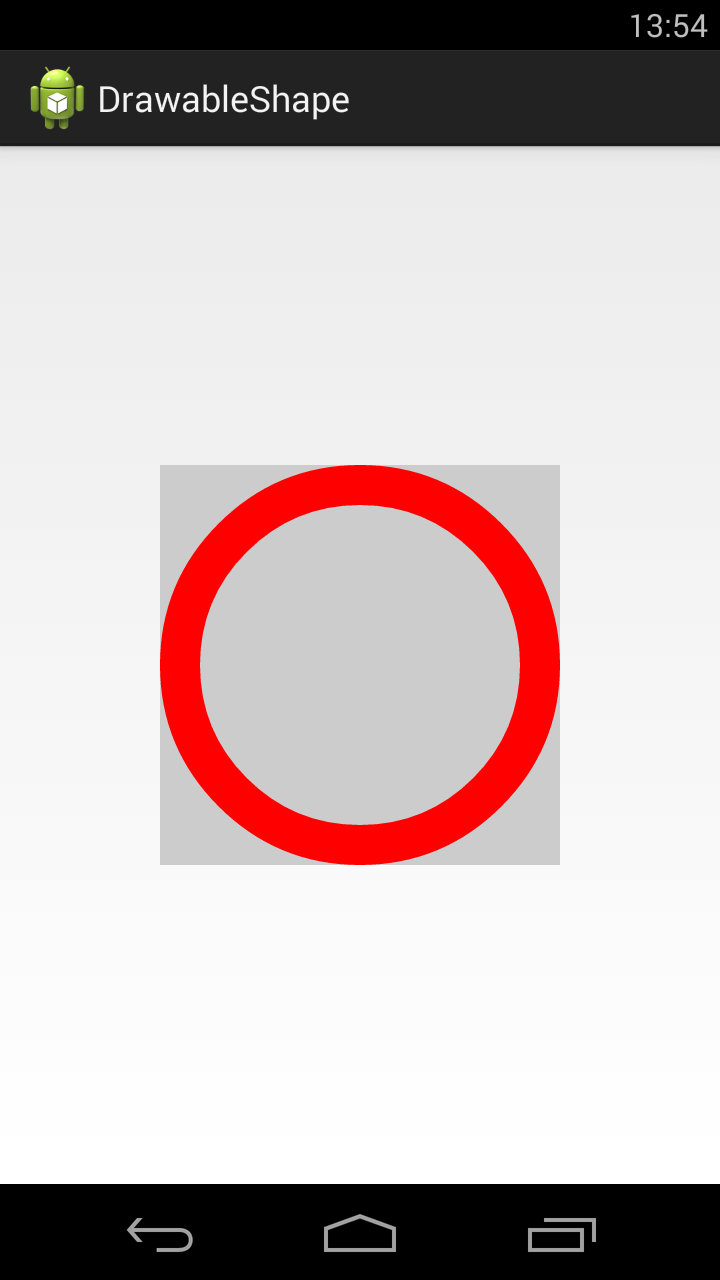
Внутренний радиус = 200 / 3 = 67. Толщина = 200 / 9 = 22. Т.е. диаметр кольца получается = 22 + 67 \* 2 + 22 = 178.

Попробуем подогнать размер кольца под все выделенное ему пространство.

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:innerRadiusRatio=*"2.5"*  
 android:shape=*"ring"*  
 android:thicknessRatio=*"10"*  
 android:useLevel=*"false"*>  
<solid  
 android:color=*"#FF0000"*>  
</solid>  
<size  
 android:height=*"200dp"*  
 android:width=*"200dp"*>  
</size>  
</shape>

Теперь радиус будет равен 200 / 2.5 = 80, а толщина = 200 / 10 = 20. Диаметр кольца =  20 + 80 \* 2 + 20 = 200.

Это видно и на скрине. Кольцо теперь по размеру равно ImageView, т.е. = 200.



Атрибуты абсолютных значений (innerRadius и thickness) имеют приоритет перед относительными (innerRadiusRatio и thicknessRatio).

### GradientDrawable

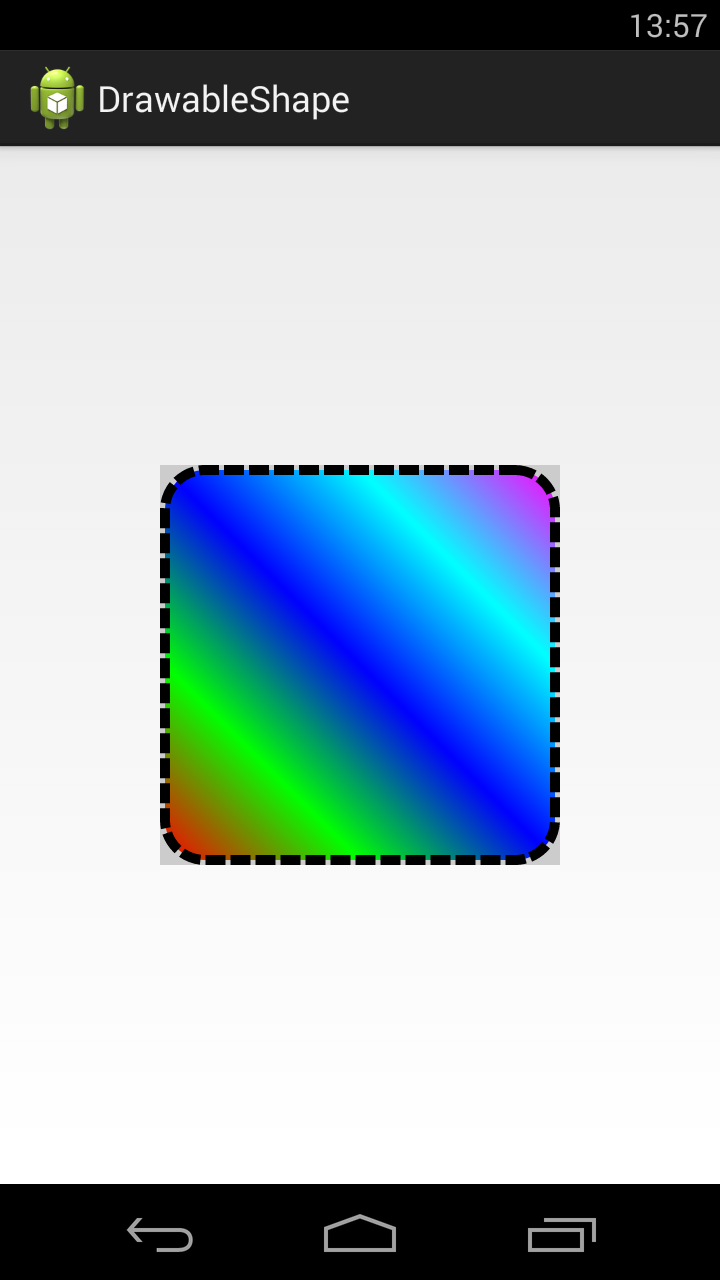
Хоть корневой тег и называется shape, но когда система его распарсит, она создает не ShapeDrawable, а GradientDrawable.

Также, этот объект мы можем сами создать программно.

Перепишем метод **setDrawable**:

**private void** setDrawable() {  
    GradientDrawable drawable = **new** GradientDrawable(GradientDrawable.Orientation.BL\_TR,   
        **new int**[] { Color.RED, Color.GREEN, Color.BLUE, Color.CYAN, Color.MAGENTA });  
    drawable.setShape(GradientDrawable.RECTANGLE);  
    drawable.setGradientType(GradientDrawable.LINEAR\_GRADIENT);  
    drawable.setCornerRadius(40);  
    drawable.setStroke(10, Color.BLACK, 20, 5);  
    imageView.setImageDrawable(drawable);  
  }

Методы set\* позволяют нам установить почти все те же параметры, что и в xml-файле.



Но есть и отличия. Например, я не нашел как для GradientDrawable установить значения размеров для кольца. С другой стороны, мы в GradientDrawable можем указать больше трех цветов для градиента.

Т.е. в основном -  xml- и java-создание равноценны, но есть некоторые нюансы.