**[Урок 163. Графика. Drawable. Bitmap, Layer List, State List.](http://startandroid.ru/ru/uroki/vse-uroki-spiskom/381-urok-163-grafika-drawable-bitmap-layer-list-state-list.html)**

В этом уроке:

- изучаем drawable теги: <bitmap>, <layer-list>, <selector>

Продолжаем разбирать типы Drawable, которые можно описывать с помощью XML-разметки. Проектов в этом уроке создавать не будем. Я просто буду в своем проекте создавать XML-файлы в папке drawable и ставить их фоном для View. А в тексте урока приведу код и скрины. Иногда буду вешать дополнительно серый фон, чтобы был виден реальный размер View.

Чтобы программно добраться до Drawable, который вы для View повесили как фон, надо просто вызвать метод [getBackground](http://developer.android.com/reference/android/view/View.html" \l "getBackground()" \t "_blank).

### Bitmap

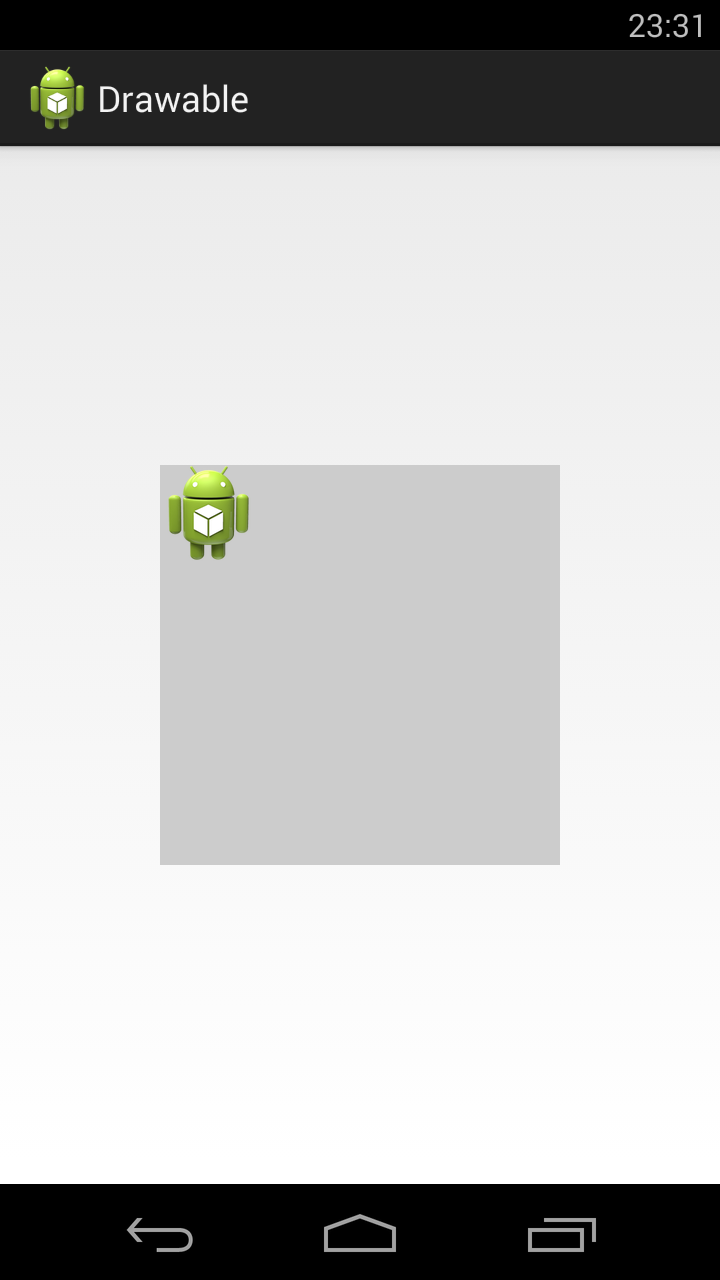
Тег [<bitmap>](http://developer.android.com/guide/topics/resources/drawable-resource.html#Bitmap) позволяет получить Drawable обертку для Bitmap. У тега есть несколько атрибутов.

В атрибуте **src** указываем нужный нам файл-изображение.

Атрибут **gravity** указывает, как bitmap будет располагаться внутри Drawable. Можно использовать несколько значений, разделенных знаком | . Значения тут стандартные, и некоторые из них мы часто используем при работе с layout. Рассмотрим пример.

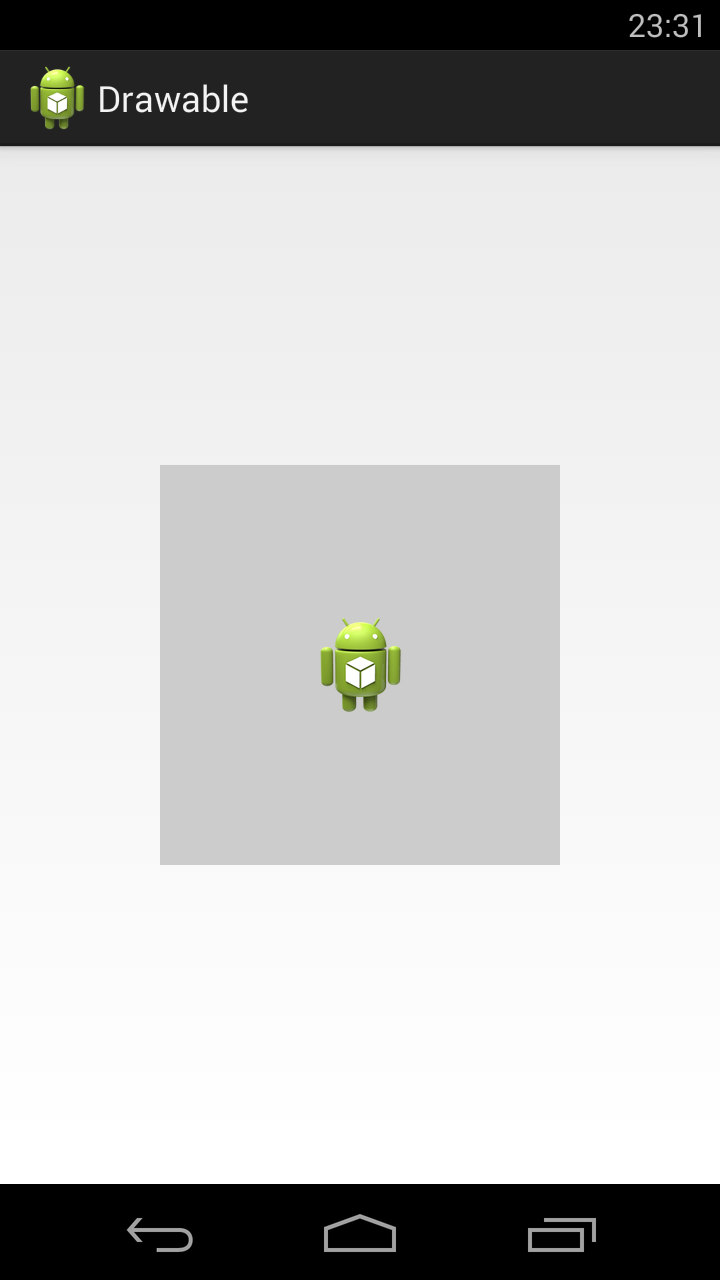
<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<bitmap  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:gravity=*"top|left"*  
 android:src=*"@drawable/ic\_launcher"*>  
</bitmap>

Значение атрибута gravity сдвигает изображение влево-вверх

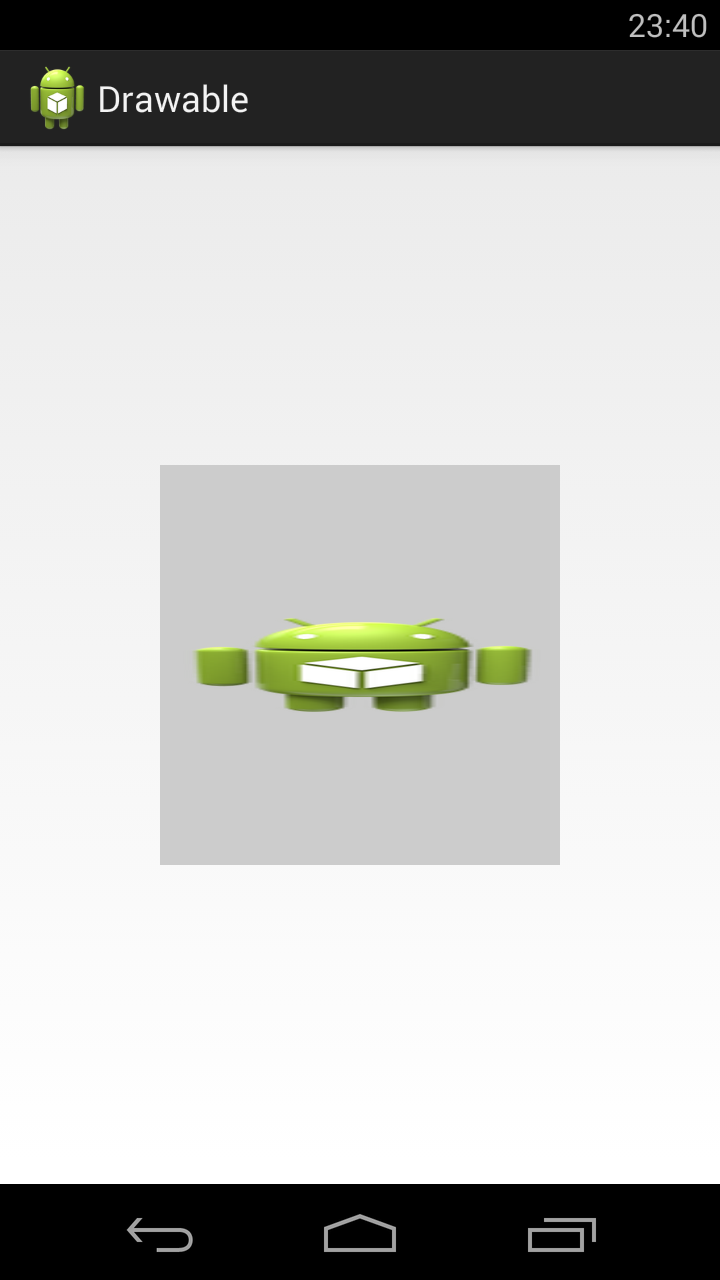


Далее ставим следующие значение атрибута gravity:

***center*** - центр



***fill\_horizontal*** - растянуть по горизонтали



***fill*** *-*растянуть (используется по умолчанию)



Насколько я понял, значения clip\_vertical и clip\_horizontal идентичны значениям fill\_vertical и fill\_horizontal в случае когда Bitmap по размеру больше, чем предоставляемое ему пространство. Т.е. clip\_vertical сожмет его по вертикали, так чтобы он влез. А clip\_horizontal - по горизонтали.

Атрибут **tileMode** - это режим «плитки». Позволяет замостить вашим изображением все доступное пространство. По умолчанию он имеет значение disabled.

Для примера я создам такой bitmap.

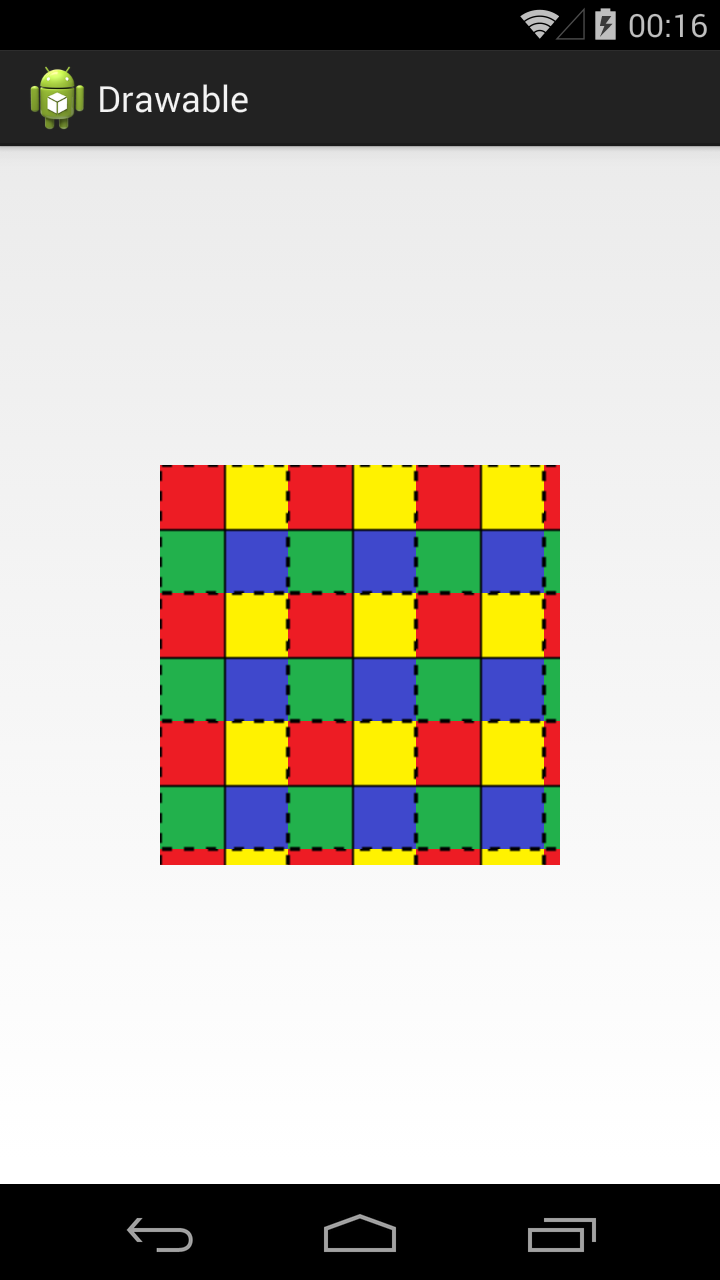


Четыре разных цвета, внутренние границы - сплошные, внешние - пунктиром.

Пример кода:

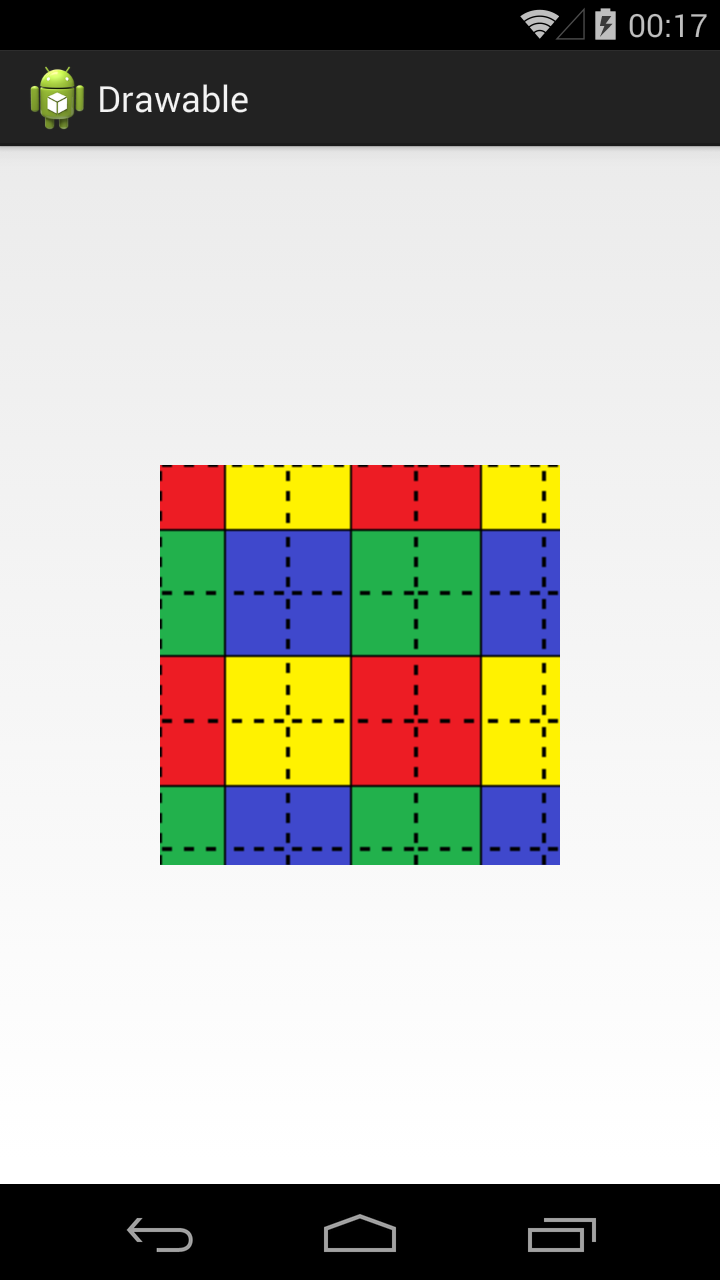
<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<bitmap  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:src=*"@drawable/tilemode"*  
 android:tileMode=*"repeat"*>  
</bitmap>

Если tileMode = ***repeat***, то Bitmap будет размножен и займет все доступное пространство

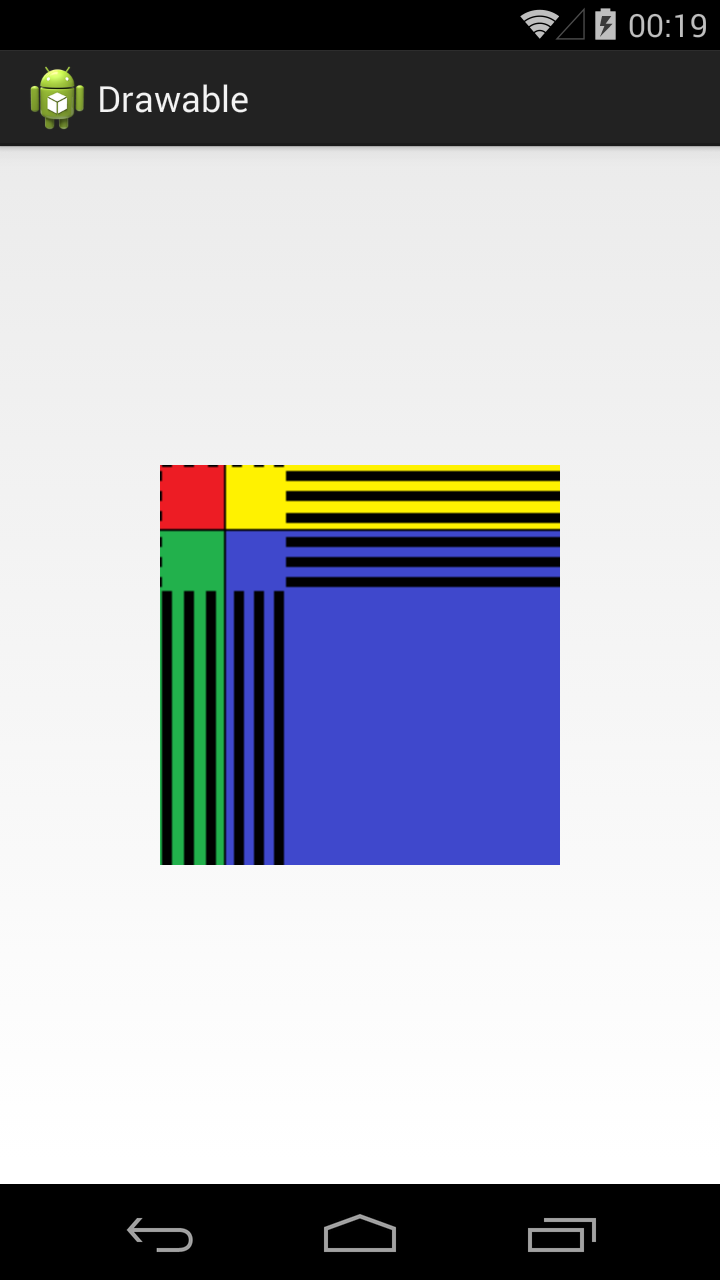


Далее меняем значение атрибута tileMode.

***mirror*** – Bitmap также будет размножен, но при этом он будет чередоваться со своим отражением



***clamp*** – растягивает края картинки на все свободное пространство



Прочие атрибуты тега <bitmap>:

**antialias** – сглаживание линий

**dither** – преобразование цветов, если текущей палитры недостаточно для отображения

**filter** – фильтр при сжатии или растягивании (пример результата использования есть в Уроке 158)

**mipMap** – использование mip-текстурирования. Про него можно почитать в [википедии](https://ru.wikipedia.org/wiki/MIP-%D1%82%D0%B5%D0%BA%D1%81%D1%82%D1%83%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5" \t "_blank). Используйте этот режим, если планируете в процессе отображения уменьшать bitmap более чем в два раза.

Мы рассмотрели XML-описание, но вы всегда можете создать этот объект и программно. Java-реализация – класс [BitmapDrawable](http://developer.android.com/reference/android/graphics/drawable/BitmapDrawable.html" \t "_blank).

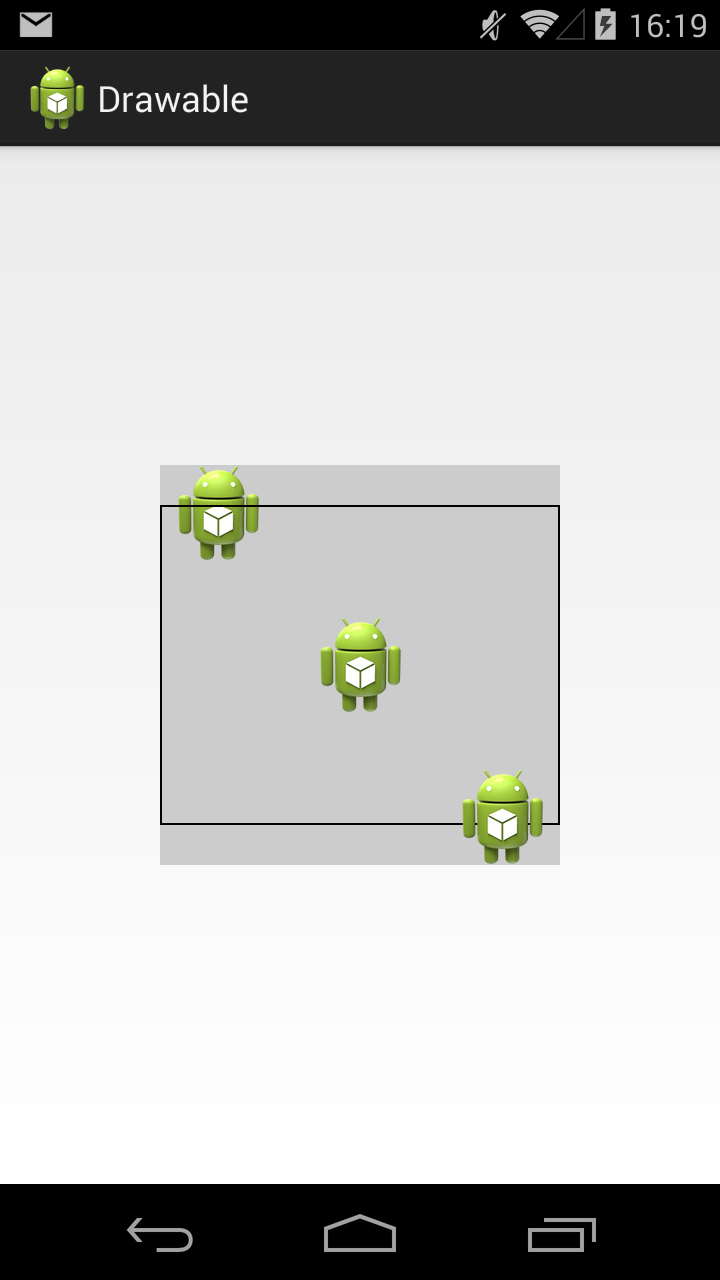
### Layer List

Мы можем описать Drawable, который будет состоять из нескольких Drawable-слоев. Для этого используется тег [<layer-list>](http://developer.android.com/guide/topics/resources/drawable-resource.html#LayerList), а внутри него теги <item>.

Пример:

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<layer-list  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*>  
<item  
 android:left=*"5dp"*>  
<bitmap  
 android:gravity=*"left|top"*  
 android:src=*"@drawable/ic\_launcher"*>  
</bitmap>  
</item>  
<item>  
<bitmap  
 android:gravity=*"center"*  
 android:src=*"@drawable/ic\_launcher"*>  
</bitmap>  
</item>  
<item  
 android:id=*"@+id/someDrawable"*  
 android:bottom=*"20dp"*  
 android:top=*"20dp"*>  
<shape  
 android:shape=*"rectangle"*>  
<stroke  
 android:width=*"1dp"*  
 android:color=*"#000000"*>  
</stroke>  
</shape>  
</item>  
<item  
 android:right=*"5dp"*>  
<bitmap  
 android:gravity=*"right|bottom"*  
 android:src=*"@drawable/ic\_launcher"*>  
</bitmap>  
</item>  
</layer-list>

У нас 4 слоя. Три bitmap со стандартной иконкой и одна фигура. Атрибуты **left**, **top**, **right**, **bottom**позволяют указывать отступы. А в атрибуте **id**можно указать id этого Drawable-слоя.



Обратите внимание, что важен порядок тегов item. Каждый последующий слой рисуется поверх предыдущего. Например, на получившемся изображении видно, что прямоугольник проходит "над" верхней иконкой, но "под" нижней.

Java-реализация – класс [LayerDrawable](http://developer.android.com/reference/android/graphics/drawable/LayerDrawable.html" \t "_blank).

Мы можем в коде получать доступ к отдельным Drawable внутри LayerDrawable. Для этого сначала получаем LayerDrawable.

LayerDrawable layerDrawable = (LayerDrawable) view.getBackground();

А затем вызываем метод [findDrawableByLayerId(int id)](http://developer.android.com/reference/android/graphics/drawable/LayerDrawable.html" \l "findDrawableByLayerId(int)" \t "_blank) и указываем id, который вы указывали в атрибуте id тега item. На выходе получим Drawable.

Также у LayerDrawable есть еще несколько интересных методов

[getDrawable(int index)](http://developer.android.com/reference/android/graphics/drawable/LayerDrawable.html#getDrawable(int)) - возвращает Drawable по индексу, а не по id

[getId(int index)](http://developer.android.com/reference/android/graphics/drawable/LayerDrawable.html#getId(int)) - возвращает id по индексу

[getNumberOfLayers()](http://developer.android.com/reference/android/graphics/drawable/LayerDrawable.html#getNumberOfLayers()) - возвращает кол-во Drawable-слоев

[setDrawableByLayerId(int id, Drawable drawable)](http://developer.android.com/reference/android/graphics/drawable/LayerDrawable.html#setDrawableByLayerId(int, android.graphics.drawable.Drawable)) - заменяет Drawable по id слоя (протестить)

### State List

Тег [<selector>](http://developer.android.com/guide/topics/resources/drawable-resource.html#StateList) позволяет отображать Drawable в зависимости от состояния View. Возможные состояние View можно посмотреть в [хелпе](http://developer.android.com/guide/topics/resources/drawable-resource.html" \l "StateList" \t "_blank). Рассмотрим пример с двумя из них: checked и pressed. На экране будет ToogleButton. Эта кнопка переходит в состояние checked и обратно, если на нее нажимать. А во время нажатия, пока палец касается экрана, кнопка находится в состоянии pressed.

State List позволит нам использовать три разных Drawable для отображения кнопки в трех состояниях: обычное, checked, pressed. Для этого создадим три файла в папке drawable.

**button.xml**:

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"rectangle"*>  
<solid  
 android:color=*"@android:color/darker\_gray"*>  
</solid>  
</shape>

Прямоугольник темно-серого цвета. Этот Drawable будем отображать в обычном состоянии кнопки.

**button\_pressed.xml**:

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"rectangle"*>  
<solid  
 android:color=*"@android:color/holo\_blue\_dark"*>  
</solid>  
</shape>

Прямоугольник темно-синего цвета. Этот Drawable будем отображать в нажатом состоянии кнопки.

**button\_checked.xml**:

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<shape  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*  
 android:shape=*"rectangle"*>  
<solid  
 android:color=*"@android:color/holo\_blue\_bright"*>  
</solid>  
</shape>

Прямоугольник светло-синего цвета. Этот Drawable будем отображать когда кнопка находится в состоянии checked.

И еще один файл, **button\_selector.xml**:

<?xml version=*"1.0"* encoding=*"utf-8"*?>  
<selector  
 xmlns:android=*"http://schemas.android.com/apk/res/android"*>  
<item  
 android:drawable=*"@drawable/button\_pressed"*  
 android:state\_pressed=*"true"*>  
</item>  
<item  
 android:drawable=*"@drawable/button\_checked"*  
 android:state\_checked=*"true"*>  
</item>  
<item  
 android:drawable=*"@drawable/button"*>  
</item>  
</selector>

Этот последний Drawable является селектором. В нем мы используем теги item, в которых указываем для какого состояния какой Drawable использовать

В первом item мы указали state\_pressed=true, а значит этот item будет выбран системой когда кнопка будет в состоянии pressed. И экране мы увидим Drawable из этого item, т.е. button\_pressed.

В втором item мы указали state\_checked=true, а значит этот item будет выбран системой когда кнопка будет в состоянии checked. И экране мы увидим button\_checked.

В третьем item мы не указали никакого состояния, этот item будет выбран при обычном состоянии кнопки. И экране мы увидим button.

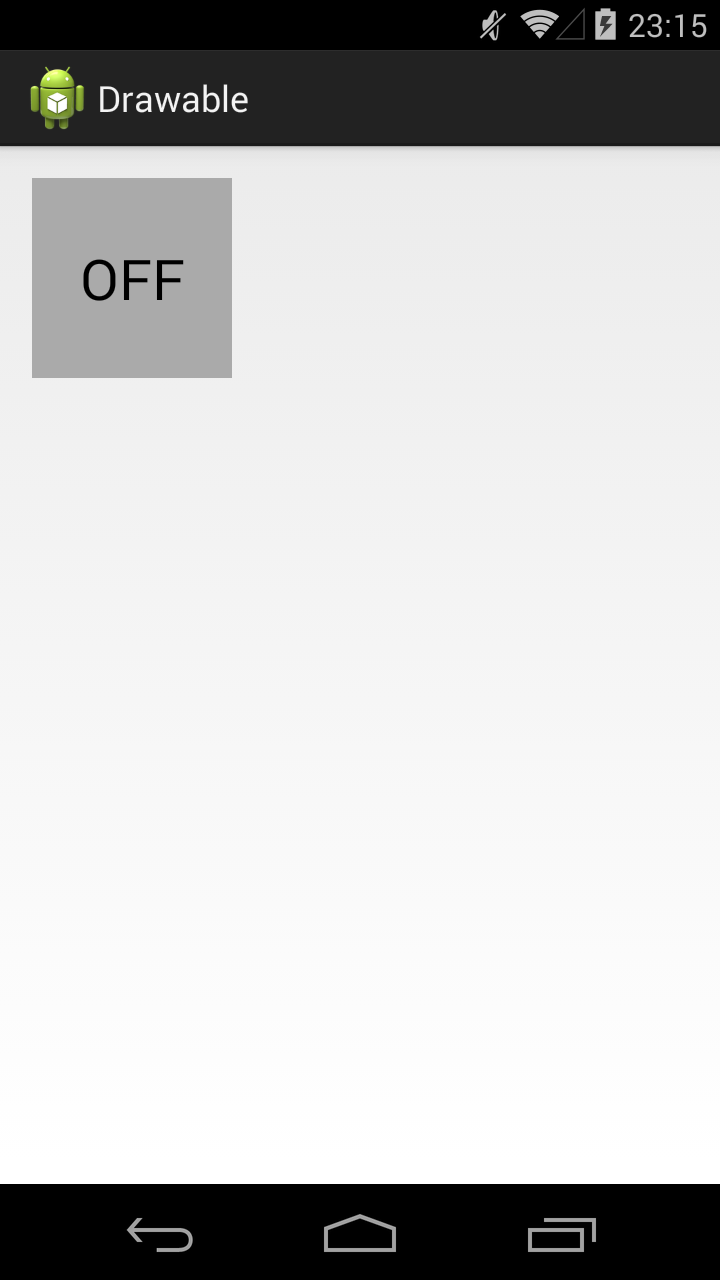
Учтите, что здесь важен порядок расположения item внутри selector. Т.е. система идет по ним по порядку и выбирает первый подходящий. Если вы третий item, который без явного указания состояния, поставите первым, то система всегда будет останавливаться на нем.

Состояния можно комбинировать, т.е. в одном item вы можете указать несколько разных состояний.

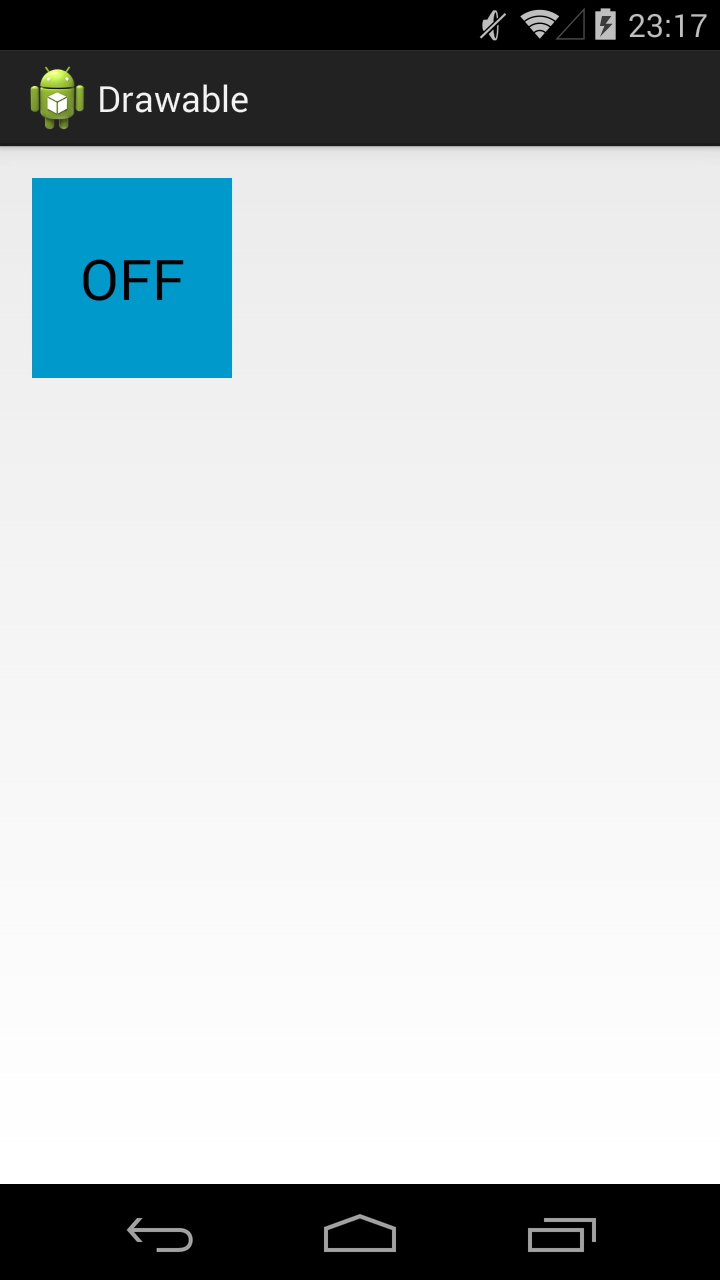
Ставим этот Drawable, как фон для ToogleButton:

<ToggleButton  
 android:id=*"@+id/toggleButton1"*  
 android:layout\_width=*"100dp"*  
 android:layout\_height=*"100dp"*  
 android:background=*"@drawable/button\_selector"*  
 android:textSize=*"28sp"*>  
</ToggleButton>

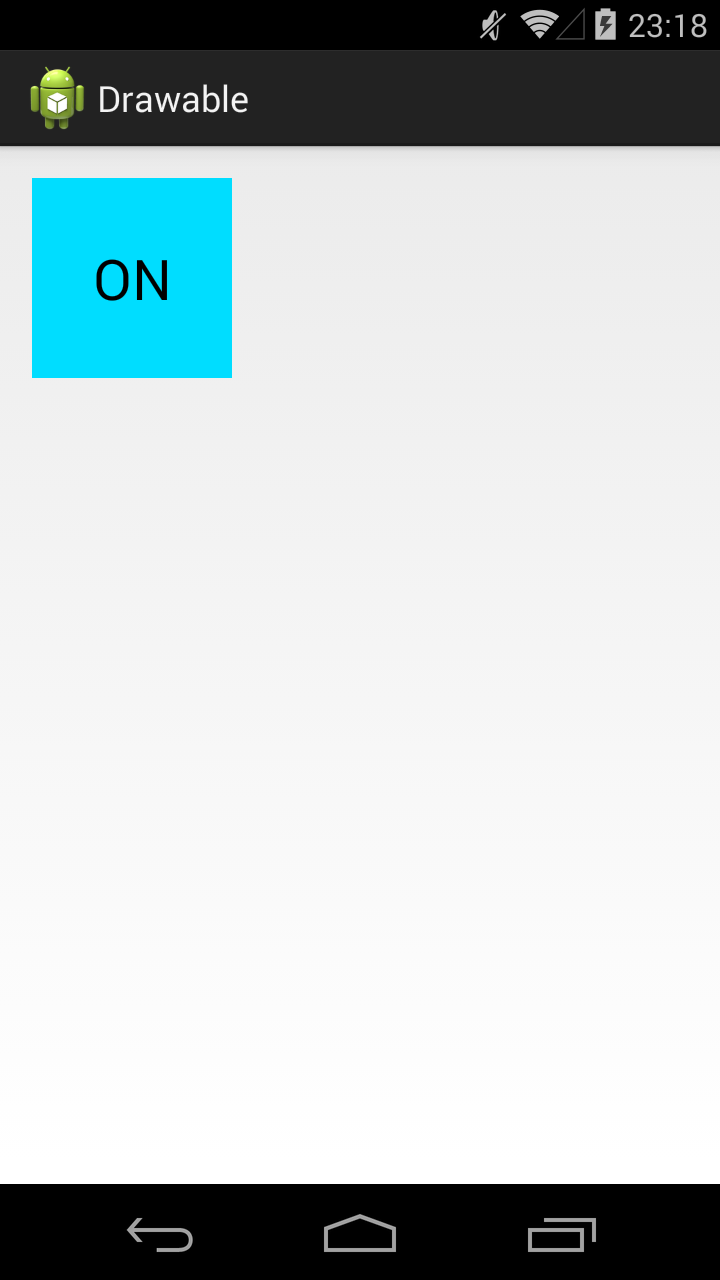
В результате, сначала видим обычное состояние



Нажимаем и держим, т.е. состояние pressed



Отпускаем – включился checked



Еще раз нажмем-отпустим - выключится checked и будет снова обычное состояние. Для каждого состояния отображается свой Drawable.

Java-реализация – класс [StateListDrawable](http://developer.android.com/reference/android/graphics/drawable/StateListDrawable.html" \t "_blank).

У View, кстати, есть методы, которые позволяют программно управлять состоянием. Это, например: [setPressed](http://developer.android.com/reference/android/view/View.html" \l "setPressed(boolean)" \t "_blank) и [setSelected](http://developer.android.com/reference/android/view/View.html" \l "setSelected(boolean)" \t "_blank).