**[Урок 147. Рисование. Region](http://startandroid.ru/ru/uroki/vse-uroki-spiskom/325-urok-147-risovanie-region.html)**

В этом уроке:

- используем Region

[Region](http://developer.android.com/reference/android/graphics/Region.html) - это объект, который позволяет нам совмещать несколько фигур в одну, используя различные режимы: объединение, пересечение и пр.  На словах трудновато будет объяснить подробно, поэтому давайте пример смотреть.

Создадим проект:

**Project name**: P1471\_Region  
**Build Target**: Android 2.3.3   
**Application name**: Region  
**Package name**: ru.startandroid.develop.p1471region  
**Create Activity**: MainActivity

**MainActivity.java**:

**package** ru.startandroid.develop.p1471region;  
  
**import** android.app.Activity;  
**import** android.content.Context;  
**import** android.graphics.Canvas;  
**import** android.graphics.Color;  
**import** android.graphics.Paint;  
**import** android.graphics.Path;  
**import** android.graphics.Rect;  
**import** android.graphics.Region;  
**import** android.graphics.RegionIterator;  
**import** android.os.Bundle;  
**import** android.view.View;  
  
**public class** MainActivity **extends** Activity {  
  
  @Override  
  **protected void** onCreate(Bundle savedInstanceState) {  
    **super**.onCreate(savedInstanceState);  
    setContentView(**new** DrawView(**this**));  
  }  
  
**class** DrawView **extends** View {  
      
    Paint p;  
    Rect rect1;  
    Rect rect2;  
    Region region;  
    RegionIterator iterator;  
    Path path;  
      
    Region.Op op = Region.Op.UNION;  
      
  
    **public** DrawView(Context context) {  
      **super**(context);  
      p = **new** Paint();  
      p.setStrokeWidth(3);  
        
      // прямоугольники  
      rect1 = **new** Rect(200,200,400,400);  
      rect2 = **new** Rect(300,300,500,500);  
        
      // создание региона  
      region = **new** Region();  
      region.set(rect1);  
      region.op(rect2, op);  
        
      // создание path из региона  
      path = region.getBoundaryPath();  
    }  
      
    @Override  
    **protected void** onDraw(Canvas canvas) {  
      canvas.drawARGB(80, 102, 204, 255);  
        
      // контуры прямоугольников  
      p.setStyle(Paint.Style.STROKE);  
      p.setColor(Color.BLACK);  
      canvas.drawRect(rect1, p);  
      canvas.drawRect(rect2, p);  
        
      // path  
      p.setStyle(Paint.Style.FILL);  
      p.setColor(Color.BLUE);  
      canvas.drawPath(path, p);  
        
    }  
  }  
}

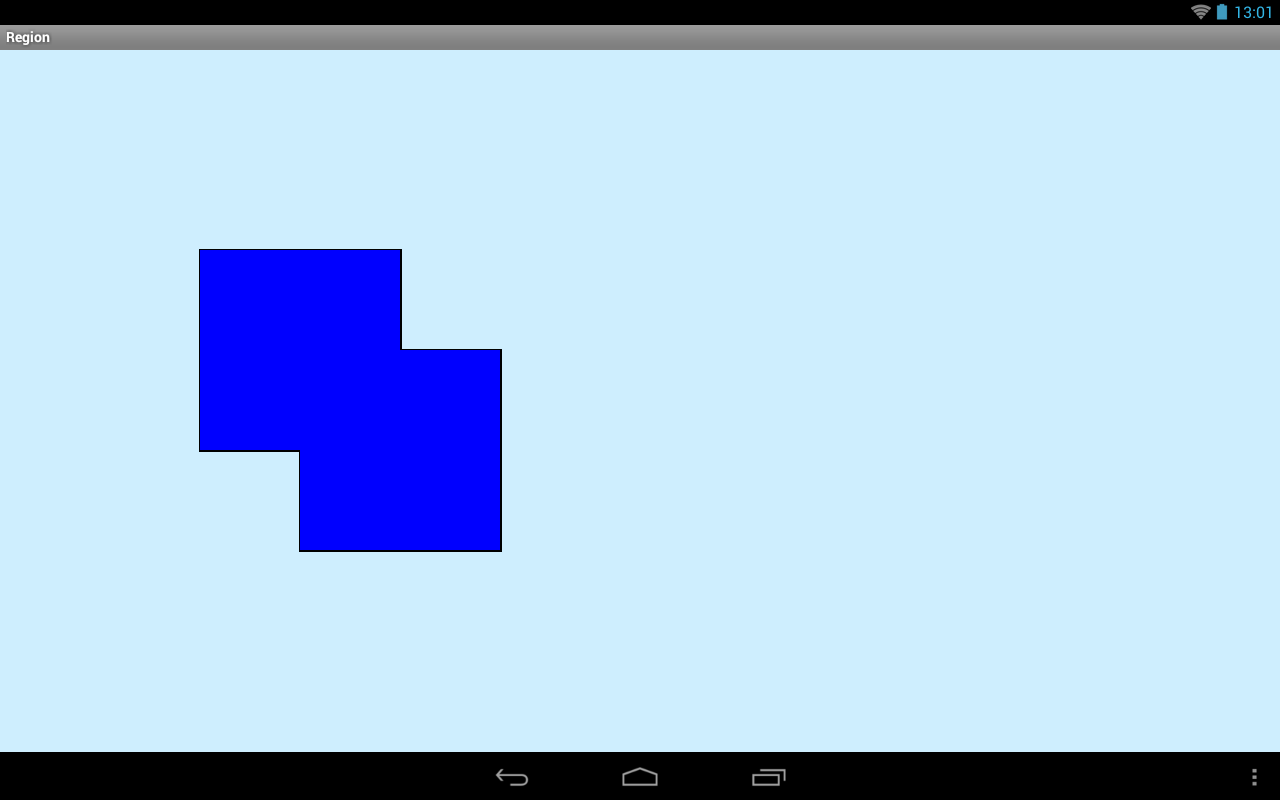
В конструкторе **DrawView**создаем объекты. У нас в примере будут участвовать два прямоугольника rect1 и rect2. Как видно по их координатам, они пересекаются. Далее создаем регион и методом [set](http://developer.android.com/reference/android/graphics/Region.html" \l "set(android.graphics.Rect)" \t "_blank) присваиваем ему первый прямоугольник (rect). Регион теперь состоит из одного прямоугольника. Чтобы добавлять к нему дополнительные прямоугольники, необходимо использовать метод [op](http://developer.android.com/reference/android/graphics/Region.html" \l "op(android.graphics.Rect, android.graphics.Region.Op)" \t "_blank). Добавляем второй прямоугольник (rect2) и при этом указываем режим Region.Op.UNION (переменная op).

Если при добавлении нового прямоугольника к региону используется режим UNION, то итоговый регион будет являться объединением области текущего региона и добавляемого прямоугольника. В нашем случае регион состоял из первого прямоугольника, а значит результатом добавления второго будет объединение областей первого и второго прямоугольника.

Далее методом [getBoundaryPath](http://developer.android.com/reference/android/graphics/Region.html" \l "getBoundaryPath()" \t "_blank) получаем итоговую область региона в объект Path, чтобы можно было нарисовать результат объединения.

В методе **onDraw**сначала рисуем черным цветом контуры прямоугольников. Затем синим цветом с заливкой рисуем path, который представляет из себя итоговую область региона.

Результат:



Видим, что регион представляет собой объединение двух прямоугольников. Объединение мы получили, т.к. использовали режим UNION.

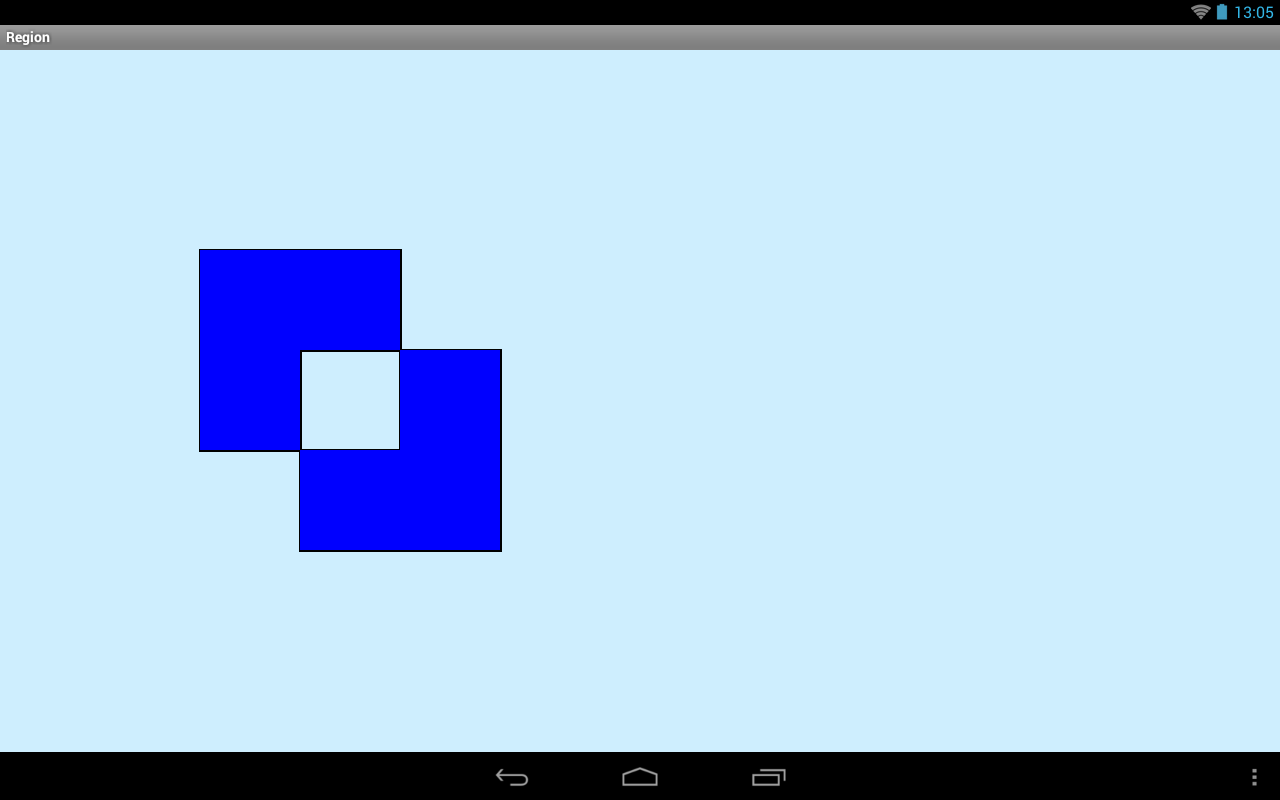
Мы рассмотрели один [режим добавления](http://developer.android.com/reference/android/graphics/Region.Op.html), а всего их 6. Смотрим остальные. Для этого в нашем коде используется переменная op:

Region.Op op = Region.Op.UNION;

Сейчас тут значение UNION. Просто меняйте его на рассматриваемые нами далее режимы.

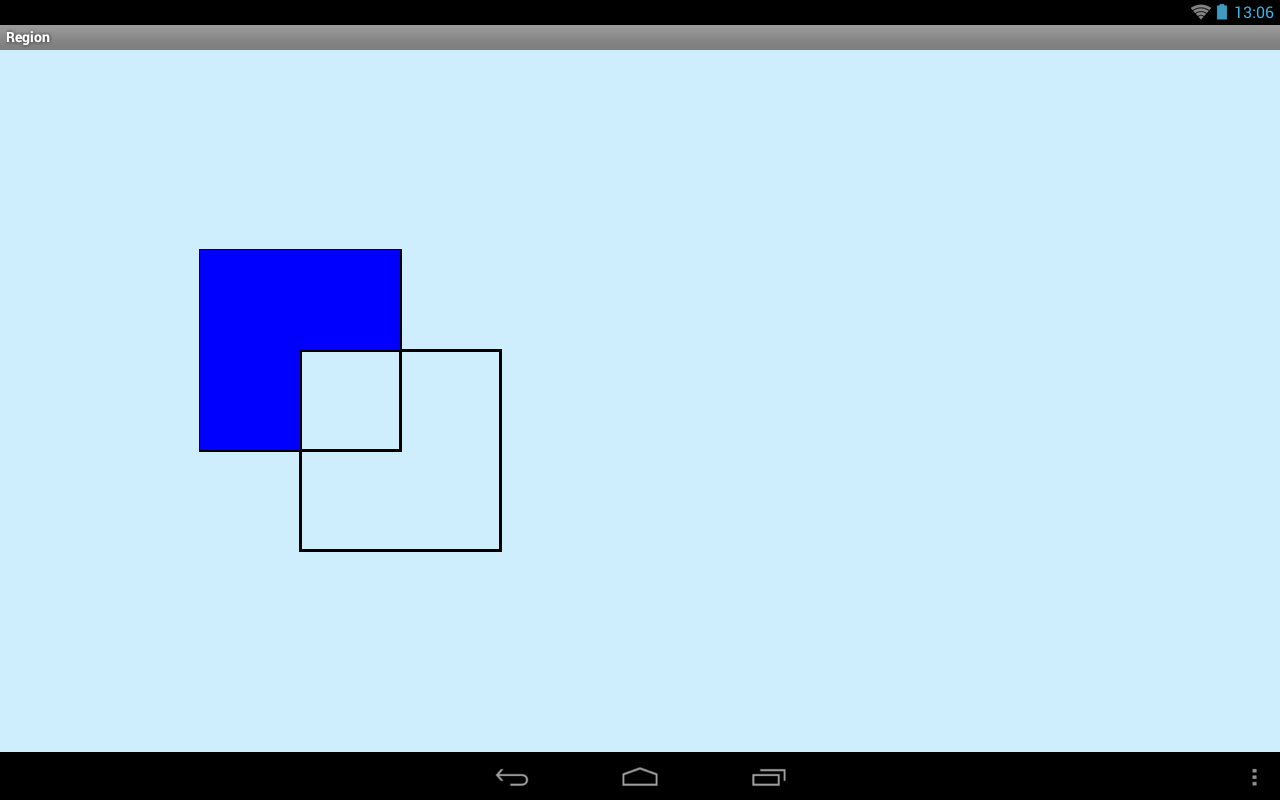
Помним, что регион изначально содержит первый прямоугольник. А второй прямоугольник мы добавляем с использованием определенного режима.

XOR



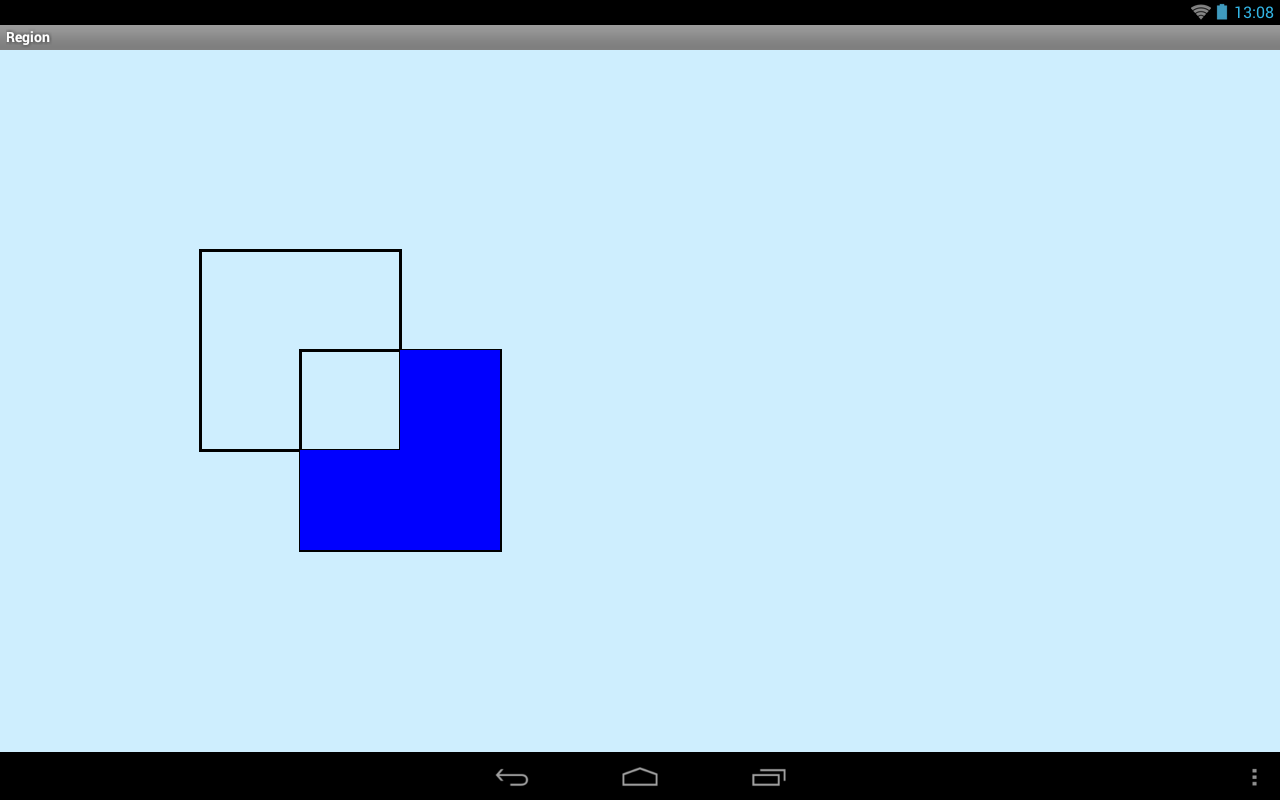
Итоговая область региона: области обоих прямоугольников за исключением их пересечения.

DIFFERENCE



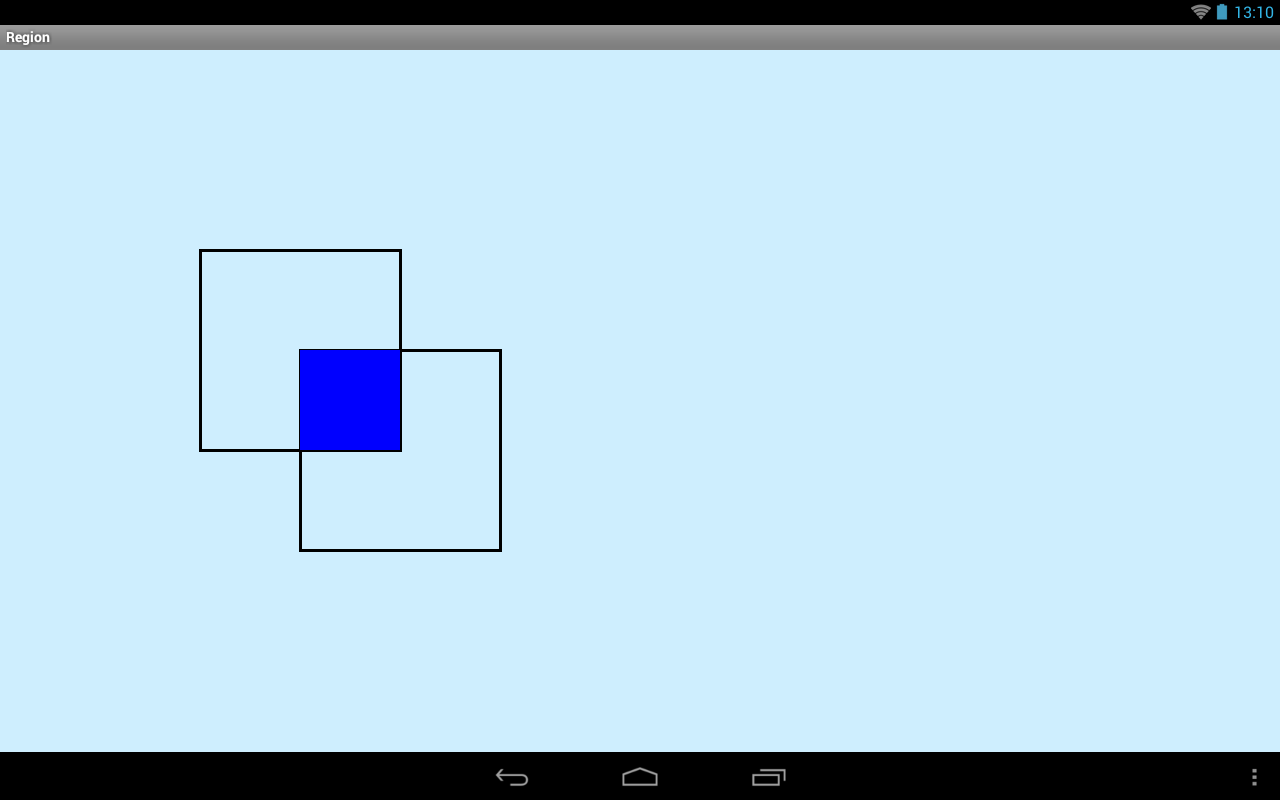
Итоговая область региона: область первого прямоугольника за исключением пересечения его со вторым.

REVERSE\_DIFFERENCE



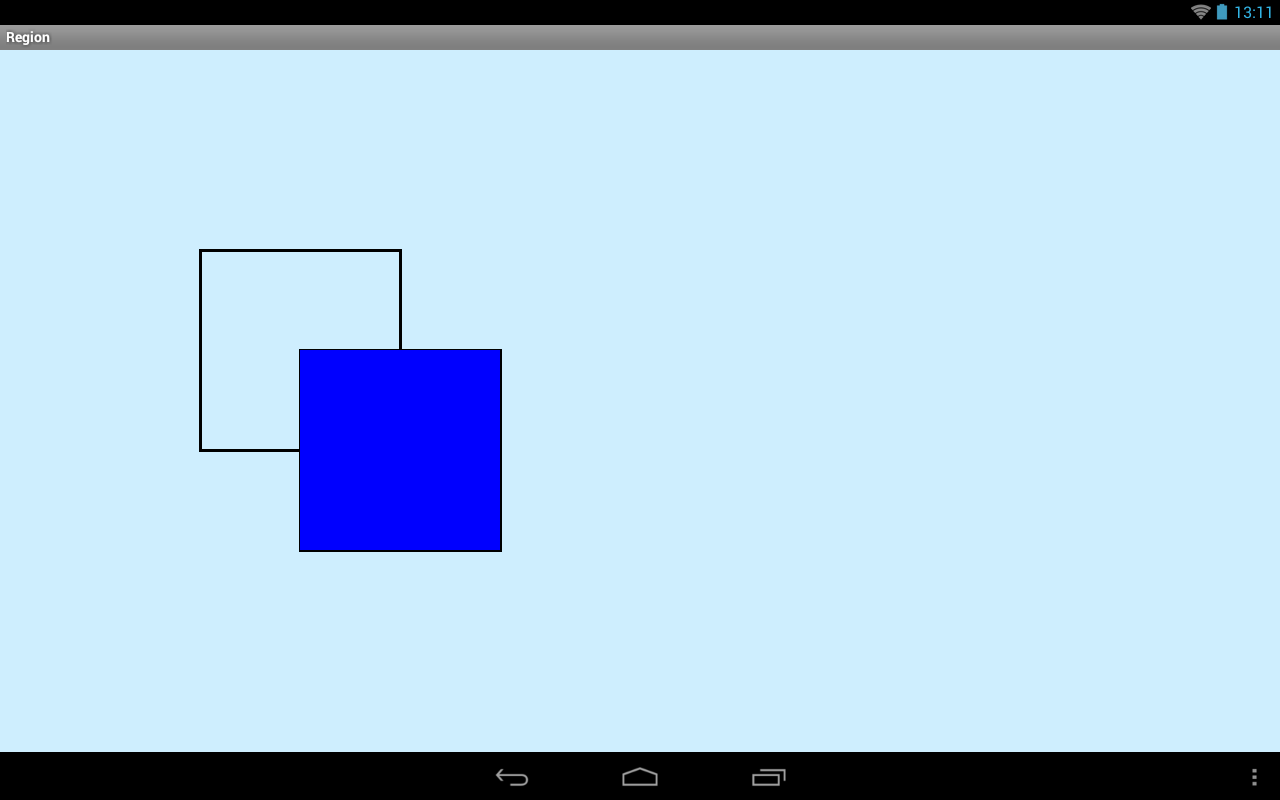
Итоговая область региона: область второго прямоугольника за исключением пересечения его с первым.

INTERSECT



Итоговая область региона: пересечение обоих прямоугольников

REPLACE



Итоговая область региона: второй прямоугольник.Т.е. содержимое региона заменилось вторым прямоугольником.

В хелпе можно увидеть, что метод op имеет несколько вариантов, но в целом смысл везде одинаков – добавление прямоугольника или целого региона к текущему региону с использованием режимов.

### RegionIterator

Итоговая область региона может быть разбита на набор непересекающихся прямоугольников. Для этого используется [RegionIterator](http://developer.android.com/reference/android/graphics/RegionIterator.html" \t "_blank) – итератор региона. При создании указываете ему регион и методом [next](http://developer.android.com/reference/android/graphics/RegionIterator.html" \l "next(android.graphics.Rect)" \t "_blank) перебираете прямоугольники, из которых состоит регион.

Ради интереса повесьте лог в цикл итератора и выведите (Rect.toShortString) координаты областей, из которых состоит регион в примерах выше. Вы увидите, как итератор разбивает регион на непересекающиеся прямоугольники.

В случае с UNION, например, лог будет следующим:

*rect = [200,200][400,300]  
rect = [200,300][500,400]  
rect = [300,400][500,500]*

### Прочие методы

Рассмотрим еще несколько полезных методов региона.

[contains](http://developer.android.com/reference/android/graphics/Region.html#contains(int, int)) – позволяет определить, содержится ли указанная точка в регионе

[getBounds](http://developer.android.com/reference/android/graphics/Region.html#getBounds()) – вернет нам прямоугольник, который является общими границами региона

[isComplex](http://developer.android.com/reference/android/graphics/Region.html#isComplex()) – вернет true, если регион состоит из более, чем одного прямоугольников. Причем имеется ввиду вовсе не количество добавленных к региону прямоугольников. Здесь речь о том, сколько прямоугольников содержит итератор региона.

[isRect](http://developer.android.com/reference/android/graphics/Region.html#isRect()) – вернет true, если итоговая область региона является единым прямоугольником

[quickContains](http://developer.android.com/reference/android/graphics/Region.html#quickContains(android.graphics.Rect)) – вернет true если регион является единым прямоугольником и содержит в себе переданный ему прямоугольник. При этом false вовсе не означает, что переданный прямоугольник обязательно НЕ содержится в этом регионе.

[quickReject](http://developer.android.com/reference/android/graphics/Region.html#quickReject(android.graphics.Rect)) – вернет true, если регион пуст или не пересекается с переданным прямоугольником/регионом. При этом false вовсе не означает, что переданный прямоугольник/регион обязательно НЕ пересекаются с текущим.

[setPath](http://developer.android.com/reference/android/graphics/Region.html#setPath(android.graphics.Path, android.graphics.Region)) – позволяет нам отсекать от переданного Path кусок, ограниченный переданным регионом. Отсеченный кусок будет итоговой областью текущего региона.

Посмотрим на примере, перепишем **DravView**:

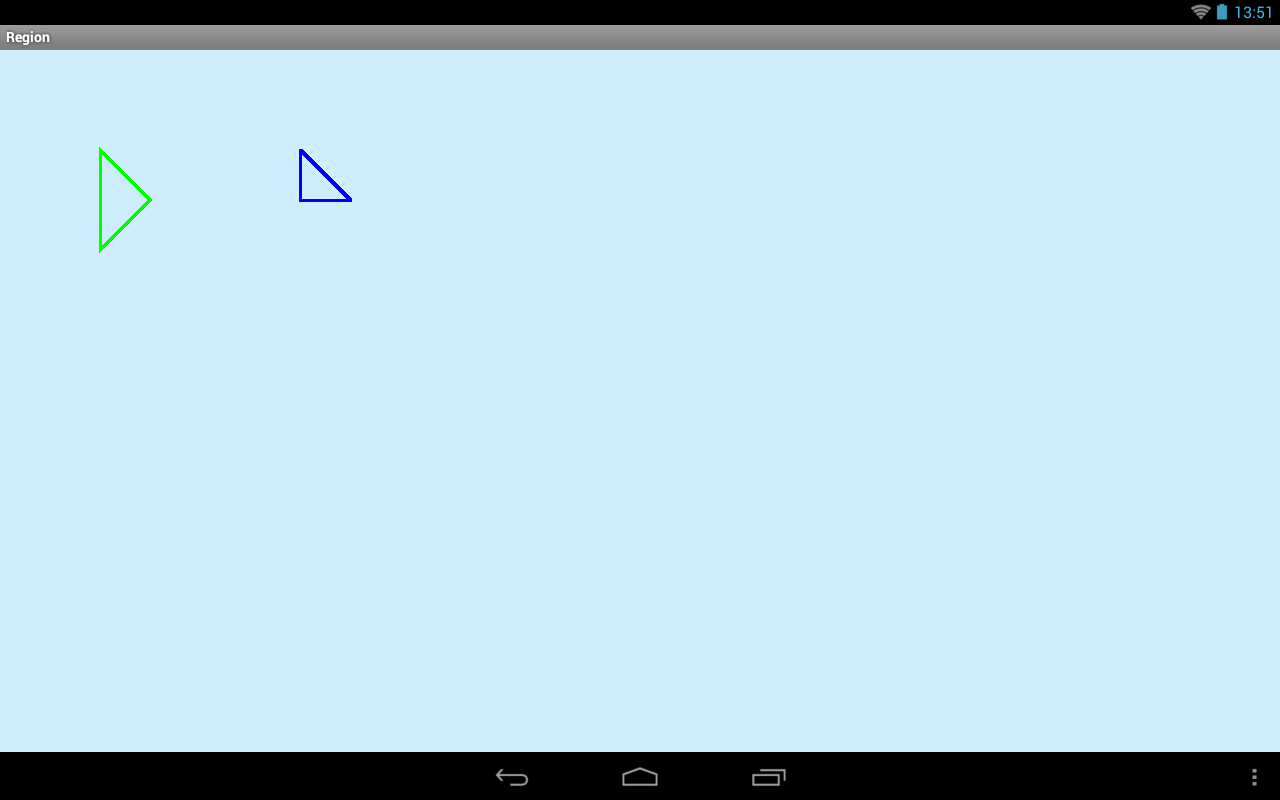
**class** DrawView **extends** View {  
  
    Paint p;  
    Region region;  
    Region clipRegion;  
    Path path;  
    Path pathDst;  
    Rect rect;  
  
    **public** DrawView(Context context) {  
      **super**(context);  
      p = **new** Paint();  
      p.setStrokeWidth(3);  
      p.setStyle(Paint.Style.STROKE);  
  
      // path, треугольник  
      path = **new** Path();  
      path.moveTo(100, 100);  
      path.lineTo(150, 150);  
      path.lineTo(100, 200);  
      path.close();  
  
      // регион из прямоугольника обрезки  
      rect = **new** Rect(100, 100, 150, 150);  
      clipRegion = **new** Region(rect);  
  
      // итоговый регион  
      region = **new** Region();  
      // отсекаем от path область clipRegion  
      region.setPath(path, clipRegion);  
      // получаем path из региона  
      pathDst = region.getBoundaryPath();  
  
    }  
  
    @Override  
    **protected void** onDraw(Canvas canvas) {  
      canvas.drawARGB(80, 102, 204, 255);  
  
      // треугольник  
      p.setColor(Color.GREEN);  
      canvas.drawPath(path, p);  
  
      canvas.translate(200, 0);  
  
      // верхняя часть треугольника  
      p.setColor(Color.BLUE);  
      canvas.drawPath(pathDst, p);  
  
    }  
  }

Все основные операции происходят в конструкторе **DrawView**. Сначала создаем path, в виде треугольника. Затем создаем прямоугольник rect, который, как видно по координатам, заключает в себе верхнюю половину треугольника. Именно эту часть мы сейчас и будет отделять от path. Создаем регион clipRegion, итоговой областью которого будет являться rect.

Далее создаем новый регион и выполняем для него метод setPath. На вход передаем path, от которого надо отделить часть, и регион в пределах которого, находится эта отделяемая часть. В итоге переменная region у нас теперь содержит верхнюю отделенную часть треугольника. Формируем из него новый Path в переменную pathDst методом getBoundaryPath.

В **onDraw** выводим зеленым цветом изначальный треугольник, а синим цветом рисуем его отрезанный верхний кусок.

Результат:



Я ради интереса создал итератор для верхней половины треугольника и вот, что получил

[300,100][301,101][300,101][302,102][300,102][303,103]

...

[300,147][348,148][300,148][349,149][300,149][350,150]

Видно, что регион разбил треугольник на множество прямоугольников с высотой = 1.

В общем, регион - штука специфическая, и для некоторых операций - незаменимая. Мне, например, совсем недавно он пригодился, чтобы для картинки сделать карту изображений.