**[Урок 146. Рисование. Canvas-преобразования. Методы save и restore.](http://startandroid.ru/ru/uroki/vse-uroki-spiskom/322-urok-146-risovanie-canvas-preobrazovanija.html)**

В этом уроке:

- используем матрицу канвы для преобразований

Канва имеет свою матрицу, которая будет срабатывать для любого объекта, который вы собираетесь нарисовать. Методы настройки этой матрицы нам уже известны из Урока 144:

**- translate**(перемещение)  
**- scale**(изменение размера)  
**- rotate** (поворот)  
**- skew** (наклон)

У канвы эти методы являются pre-методами. Т.е. помещают преобразование в начало матрицы, сохраняя остальные.

Рассмотрим это на примере.

**Преобразования**

Создадим проект:

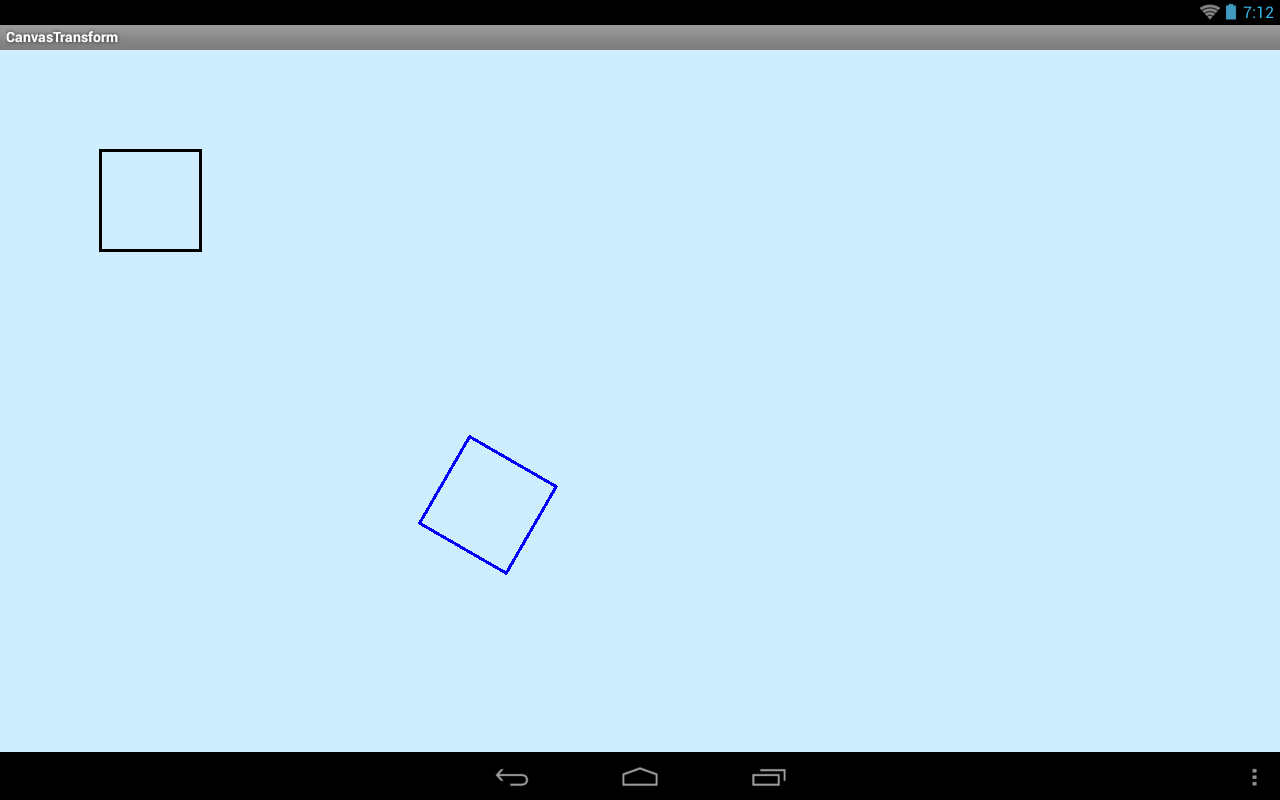
**Project name**: P1461\_CanvasTransform  
**Build Target**: Android 2.3.3   
**Application name**: CanvasTransform  
**Package name**: ru.startandroid.develop.p1461canvastransform  
**Create Activity**: MainActivity

**MainActivity.java**:

**package** ru.startandroid.develop.p1461canvastransform;  
  
**import** android.app.Activity;  
**import** android.content.Context;  
**import** android.graphics.Canvas;  
**import** android.graphics.Color;  
**import** android.graphics.Matrix;  
**import** android.graphics.Paint;  
**import** android.graphics.Path;  
**import** android.graphics.RectF;  
**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;  
    Matrix matrix;  
    RectF rectf;  
    Path path;  
  
    **public** DrawView(Context context) {  
      **super**(context);  
      p = **new** Paint();  
      p.setStrokeWidth(3);  
      p.setStyle(Paint.Style.STROKE);  
      rectf = **new** RectF(100, 100, 200, 200);  
      matrix = **new** Matrix();  
      path = **new** Path();  
    }  
  
    @Override  
    **protected void** onDraw(Canvas canvas) {  
      canvas.drawARGB(80, 102, 204, 255);  
        
      // квадрат  
      path.reset();  
      path.addRect(rectf, Path.Direction.CW);  
      p.setColor(Color.BLACK);  
      canvas.drawPath(path, p);  
        
      // преобразованный квадрат  
      matrix.reset();  
      matrix.preRotate(30);  
      matrix.preTranslate(500, 0);  
      path.transform(matrix);  
      p.setColor(Color.BLUE);  
      canvas.drawPath(path, p);  
    }  
  }  
}

Рисуем черным цветом path с прямоугольником. Затем настраиваем матрицу на поворот на 30 градусов относительно точки (0,0) (т.к. не указана иная) и на перемещение на 500 вправо. Т.к. используем методы pre, то сначала будет перемещение, потом поворот. Преобразуем path и выводим синим цветом.

Результат:



Попробуем сделать то же самое, с помощью канвы.

Перепишем **onDraw**:

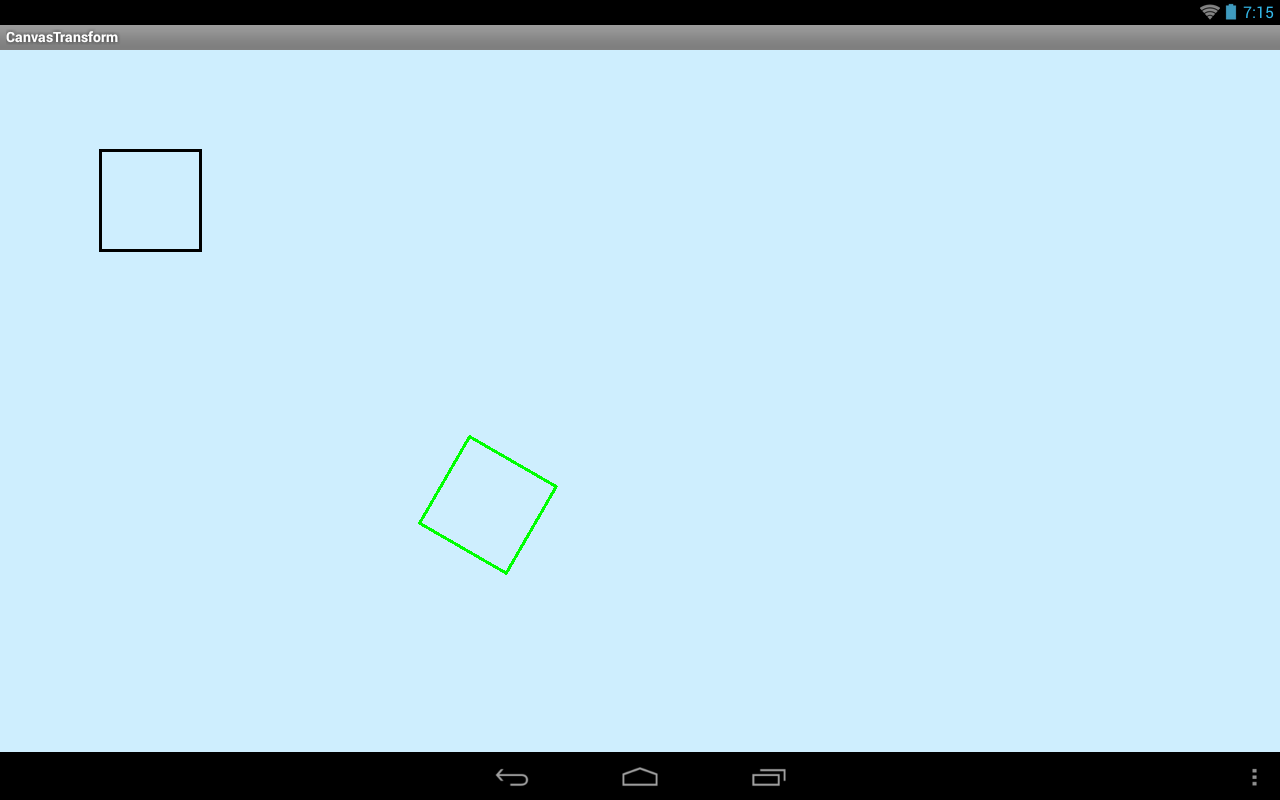
    @Override  
    **protected void** onDraw(Canvas canvas) {  
      canvas.drawARGB(80, 102, 204, 255);  
        
      // квадрат  
      p.setColor(Color.BLACK);  
      canvas.drawRect(rectf, p);  
        
      // квадрат на канве с преобразованиями  
      canvas.rotate(30);  
      canvas.translate(500, 000);  
      p.setColor(Color.GREEN);  
      canvas.drawRect(rectf, p);  
        
    }

Сначала выводим прямоугольник rectf черным цветом. Затем настраиваем матрицу канвы. Задаем те же преобразования, в том же порядке. Т.к. эти методы являются аналогами pre-методов, то сначала выполнится перемещение, затем поворот.

Матрица канвы настроена, теперь все объекты, которые мы будем рисовать на канве, будут преобразованы согласно ее матрице. Нарисуем тот же прямоугольник rectf зеленым цветом.

Заметьте, что мы никак не преобразуем rectf. Он остается с теми же координатами. Канва сама преобразует его при рисовании.

Результат:



Зеленый прямоугольник находится там же, где в прошлый раз был синий. Т.е. преобразования с помощью отдельной матрицы идентичны преобразованиям матрицы канвы.

**Сохранение и возврат состояния**

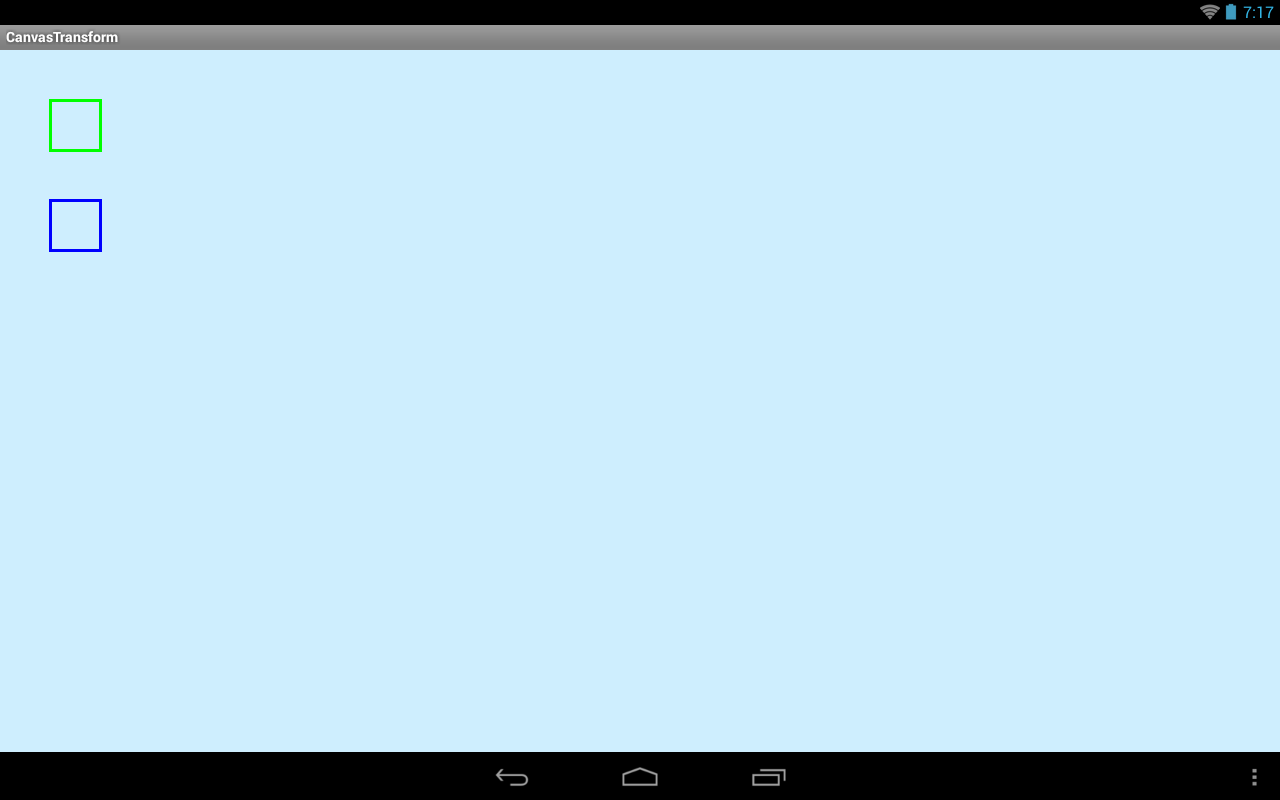
Мы можем запоминать состояние матрицы канвы и потом восстанавливать его.

Рассмотрим это на примерах. Перепишем **DrawView**:

**class** DrawView **extends** View {  
      
    Paint p;  
    RectF rectf1;  
    RectF rectf2;  
  
    **public** DrawView(Context context) {  
      **super**(context);  
      p = **new** Paint();  
      p.setStrokeWidth(3);  
      p.setStyle(Paint.Style.STROKE);  
      rectf1 = **new** RectF(50,50,100,100);  
      rectf2 = **new** RectF(50,150,100,200);  
    }  
      
    @Override  
    **protected void** onDraw(Canvas canvas) {  
      canvas.drawARGB(80, 102, 204, 255);  
        
      // зеленый квадрат  
      p.setColor(Color.GREEN);      
      canvas.drawRect(rectf1, p);  
  
      // синий квадрат  
      p.setColor(Color.BLUE);  
      canvas.drawRect(rectf2, p);  
        
    }  
  }

Нарисуем пару квадратов, первый - зеленым цветом, а второй - синим.

Без всяких преобразований они выглядят так:

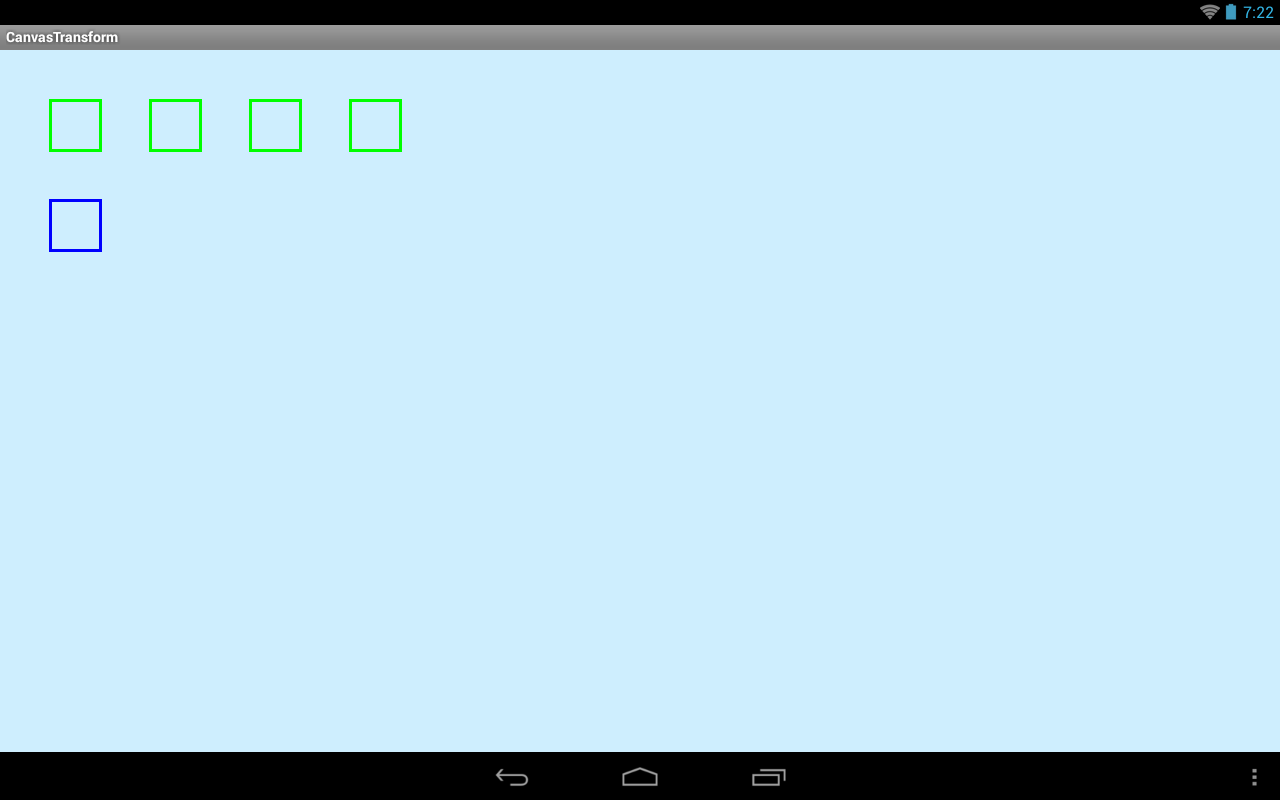


Перепишем **onDraw**:

**protected void** onDraw(Canvas canvas) {  
      canvas.drawARGB(80, 102, 204, 255);  
        
      // зеленый квадрат  
      p.setColor(Color.GREEN);      
      canvas.drawRect(rectf1, p);  
        
      // преобразования канвы  
      // и рисование зеленых квадратов  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
  
      // сброс канвы  
      canvas.restore();  
        
      // синий квадрат  
      p.setColor(Color.BLUE);  
      canvas.drawRect(rectf2, p);  
        
    }

Несколько раз настраиваем перемещение на 100 вправо и каждый раз рисуем первый квадрат зеленым цветом. Затем сбрасываем матрицу канвы методом [restore](http://developer.android.com/reference/android/graphics/Canvas.html#restore()) в изначальное состояние. И рисуем второй квадрат синим цветом.

Результат:



Видим, что первый (зеленый) квадрат рисовался со смещением согласно настройкам канвы, а метод restore все эти настройки преобразования сбросил и второй (синий) квадрат был нарисован без преобразований.

Можно настроить канву так, что метод restore будет сбрасывать ее настройки не в изначальное, а в промежуточное, сохраненное нами, состояние.

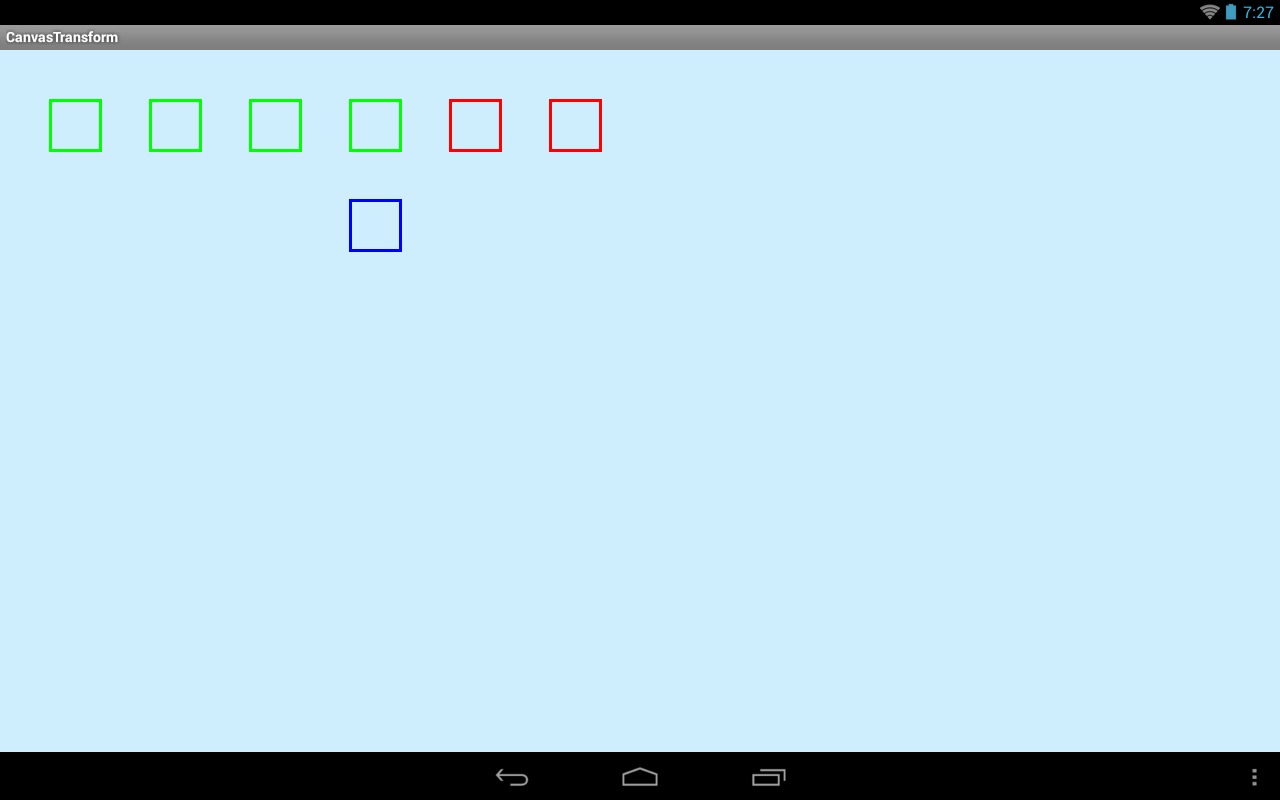
Перепишем **onDraw**:

**protected void** onDraw(Canvas canvas) {  
      canvas.drawARGB(80, 102, 204, 255);  
  
      // зеленый квадрат  
      p.setColor(Color.GREEN);  
      canvas.drawRect(rectf1, p);  
  
      // преобразования канвы  
      // и рисование зеленых квадратов  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
  
      // сохраняем настройки матрицы канвы  
      canvas.save();  
  
      // преобразования канвы  
      // и рисование красных квадратов  
      p.setColor(Color.RED);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
  
      // возврат канвы к предыдущему сохранению   
      canvas.restore();  
  
      // синий квадрат  
      p.setColor(Color.BLUE);  
      canvas.drawRect(rectf2, p);  
  
    }

Снова несколько раз настраиваем перемещение на 100 вправо и каждый раз рисуем первый квадрат зеленым цветом. Затем для канвы выполняем метод [save](http://developer.android.com/reference/android/graphics/Canvas.html#save()). Он запомнит текущие настройки матрицы. Далее еще пару раз перемещаем вправо на 100 и рисуем первый квадрат, но уже красным цветом, чтобы визуально отличить вывод до и после сохранения канвы.

Затем выполняем метод restore, который сбросит канву в состояние, которые было сохранено методом save. И рисуем второй квадрат синим цветом.

Результат:



Видим, что второй (синий) квадрат нарисован с тем состоянием канвы, которое было сохранено методом save. Метод restore вернул нас к нему из текущего состояния.

Сохранение методом save не затирает предыдущее сохраненное состояние. Т.е. вы можете несколько раз вызывать save и все эти состояния будут хранится в некоем стеке. А методом restore состояния из этого стека вытаскивать.

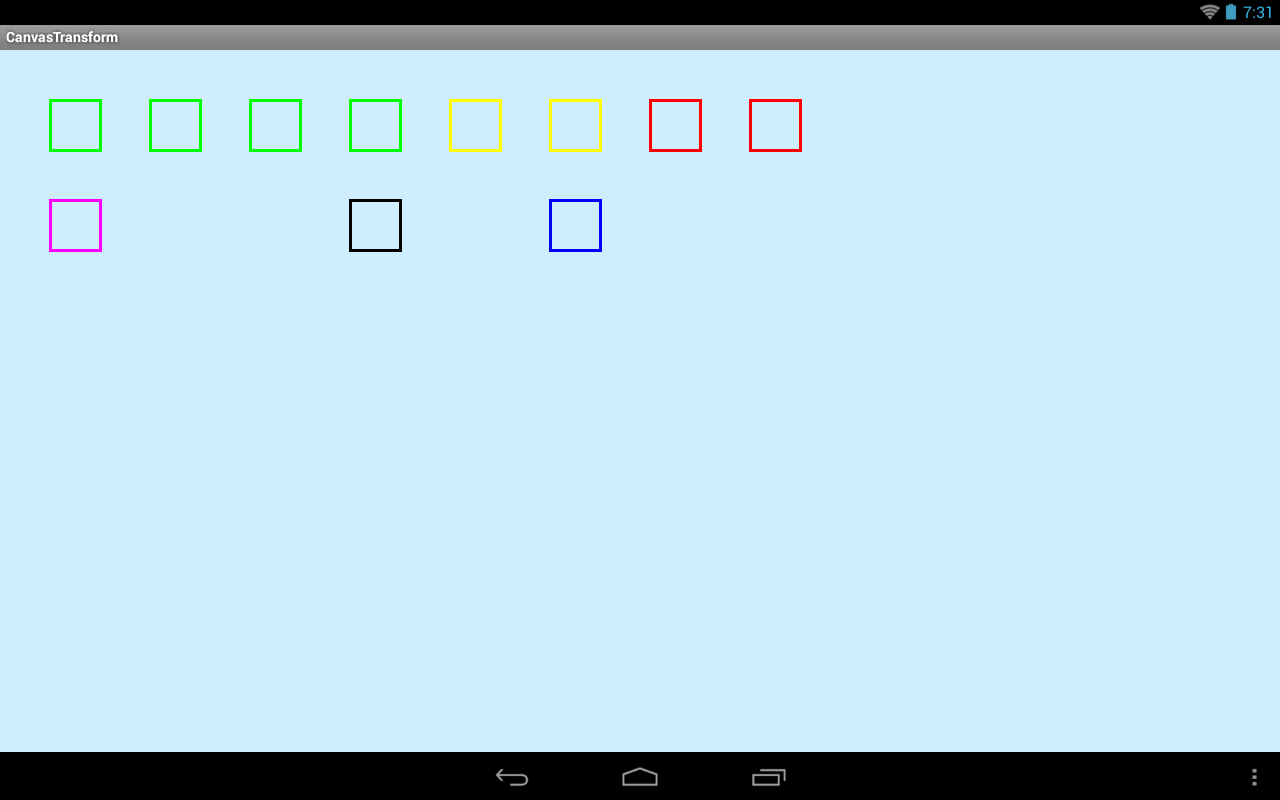
Перепишем **onDraw**:

**protected void** onDraw(Canvas canvas) {  
      canvas.drawARGB(80, 102, 204, 255);  
  
      // зеленый квадрат  
      p.setColor(Color.GREEN);  
      canvas.drawRect(rectf1, p);  
  
      // преобразования канвы  
      // и рисование зеленых квадратов  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
  
      // сохраняем настройки матрицы канвы  
      canvas.save();  
  
      // преобразования канвы  
      // и рисование желтых квадратов  
      p.setColor(Color.YELLOW);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
  
      // сохраняем настройки матрицы канвы  
      canvas.save();  
  
      // преобразования канвы  
      // и рисование красных квадратов  
      p.setColor(Color.RED);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
  
      // возврат канвы к предыдущему сохранению  
      canvas.restore();  
  
      // синий квадрат  
      p.setColor(Color.BLUE);  
      canvas.drawRect(rectf2, p);  
  
      // возврат канвы к предыдущему сохранению  
      canvas.restore();  
  
      // черный квадрат  
      p.setColor(Color.BLACK);  
      canvas.drawRect(rectf2, p);  
  
      // возврат канвы в изначальное состояние  
      canvas.restore();  
  
      // пурпурный квадрат  
      p.setColor(Color.MAGENTA);  
      canvas.drawRect(rectf2, p);  
    }

Мы несколько раз применяем перемещение на 100, рисуя первый квадрат. При этом периодически сохраняем состояние матрицы методом save, при этом меняя цвет для более удобного визуального восприятия.

Далее, мы несколько раз вызываем метод restore, который возвращает канву в сохраненные ранее состояния и рисуем второй квадрат. В итоге мы приходим к изначальному состоянию канвы.

Результат:



Метод save возвращает нам int значение. Это значение мы можем передать в метод [restoreToCount](http://developer.android.com/reference/android/graphics/Canvas.html#restoreToCount(int)) и матрица вернется к указанному состоянию, минуя остальные.

Перепишем **onDraw**:

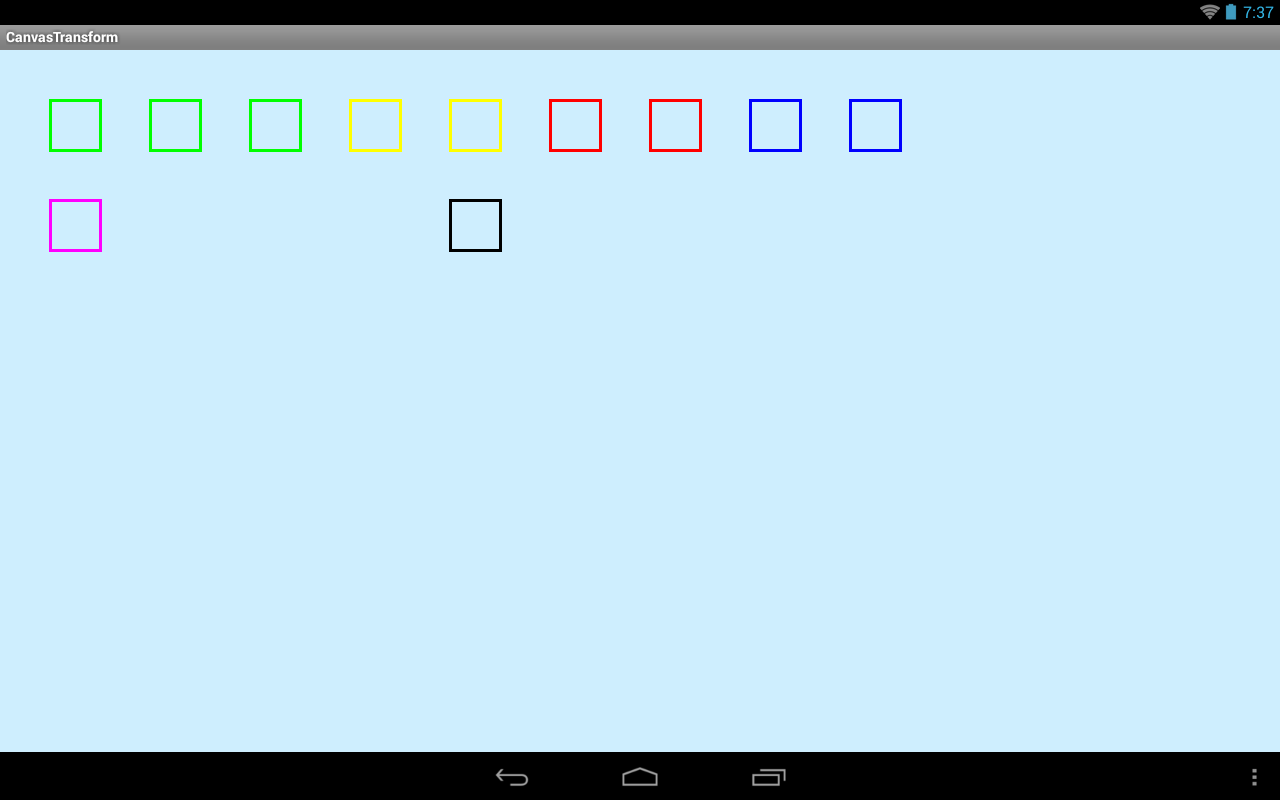
**protected void** onDraw(Canvas canvas) {  
      canvas.drawARGB(80, 102, 204, 255);  
  
      // сохраняем настройки матрицы канвы  
      // в initSave получаем значение для восстановления этого состояния  
      **int** initSave = canvas.save();  
  
      // зеленый квадрат  
      p.setColor(Color.GREEN);  
      canvas.drawRect(rectf1, p);  
  
      // преобразования канвы  
      // и рисование зеленых квадратов  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
  
      // сохраняем настройки матрицы канвы  
      canvas.save();  
  
      // преобразования канвы  
      // и рисование желтых квадратов  
      p.setColor(Color.YELLOW);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
  
      // сохраняем настройки матрицы канвы  
      // в needSave получаем значение для восстановления этого состояния  
      **int** needSave = canvas.save();  
  
      // преобразования канвы  
      // и рисование красных квадратов  
      p.setColor(Color.RED);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
  
      // сохраняем настройки матрицы канвы  
      canvas.save();  
  
      // преобразования канвы  
      // и рисование синих квадратов  
      p.setColor(Color.BLUE);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
      canvas.translate(100, 0);  
      canvas.drawRect(rectf1, p);  
  
      // возврат канвы к указанному сохранению  
      canvas.restoreToCount(needSave);  
  
      // черный квадрат  
      p.setColor(Color.BLACK);  
      canvas.drawRect(rectf2, p);  
  
      // возврат канвы к указанному сохранению  
      canvas.restoreToCount(initSave);  
  
      // пурпурный квадрат  
      p.setColor(Color.MAGENTA);  
      canvas.drawRect(rectf2, p);  
  
    }

Мы запоминаем состояние в самом начале и сохраняем значение метода save в переменную initSave. Далее мы выполняем перемещения, несколько раз сохраняем канву, и один раз пишем значение сохранения в needSave.

Затем мы возвращаемся к сохраненным состояниям, используя метод restoreToCount и переменные initSave и needSave.

В прошлом примере мы методом restore перебирали все сохраненные состояния. А здесь метод restoreToCount позволил нам вернуться сразу к необходимому состоянию.

Результат:



И напоследок еще три метода канвы

[setMatrix(Matrix matrix)](http://developer.android.com/reference/android/graphics/Canvas.html#setMatrix(android.graphics.Matrix)) – меняет матрицу канвы на указанную матрицу

[getMatrix()](http://developer.android.com/reference/android/graphics/Canvas.html#getMatrix()) – позволяет получить копию матрицы канвы

[concat(Matrix matrix)](http://developer.android.com/reference/android/graphics/Canvas.html#concat(android.graphics.Matrix)) – добавляем в начало текущей матрицы преобразования из указанной матрицы