RxJava (http://developer.alexanderklimov.ru/android/rx/)

Советы (http://developer.alexanderklimov.ru/android/tips-android.php)

Статьи (http://developer.alexanderklimov.ru/android/articles-android.php)

Книги (http://developer.alexanderklimov.ru/android/books.php)

Java (http://developer.alexanderklimov.ru/android/java/java.php)

Kotlin (http://developer.alexanderklimov.ru/android/kotlin/)

Дизайн (http://developer.alexanderklimov.ru/android/design/)

Отладка (http://developer.alexanderklimov.ru/android/debug/)

Open Source (http://developer.alexanderklimov.ru/android/opensource.php)

Полезные ресурсы (http://developer.alexanderklimov.ru/android/links.php)

Светофор

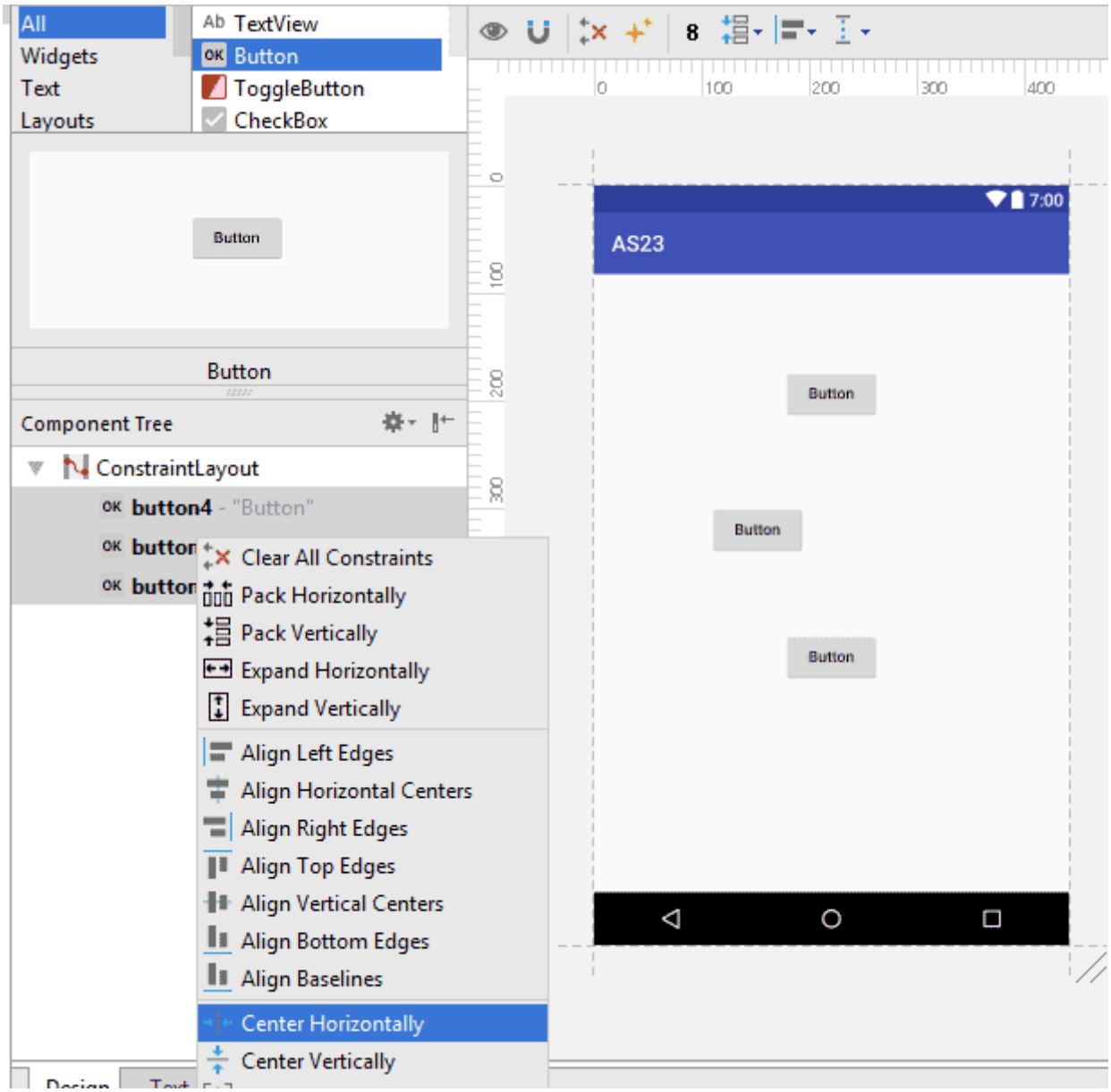
Для закрепления навыков создадим приложение чуть посложнее предыдущего, а также познакомимся с другими приёмами программирования.

В качестве примера напишем программу под условным названием «Светофор». Интерфейс программы будет выглядеть следующим образом. На красном экране расположены три кнопки и одна текстовая надпись. При нажатии кнопок фон программы будет меняться на соответствующий цвет, который

закреплён за определённой кнопкой. Я попробую вам показать решение задачи с разных сторон, чтобы вы почувствовали себя увереннее.

Первые шаги вполне очевидны. Создаём новый проект на основе "Hello, World" и перетаскиваем с панели инструментов три кнопки. Когда будете размещать кнопку, попробуйте потаскать её по разным позициям внутри экрана активности. Вы будете замечать различные всплывающие линии, подсказывающие о стандартных отступах от края или о центральной оси по вертикали и горизонтали. Это удобно, когда вы точно знаете, где нужно разместить один компонент.

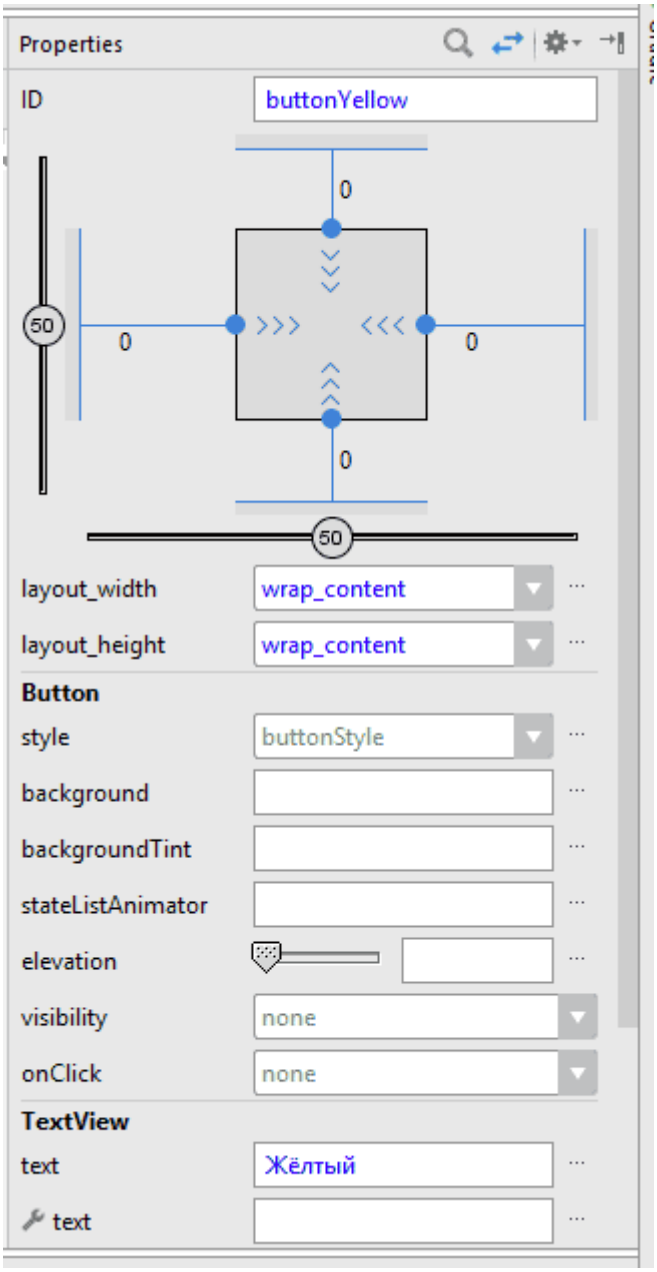
В нашем случае мы поступим иначе. Сначала просто перетащим три кнопки на экран, не думая о позиционировании. Далее в окне **Component Tree** выделяем три кнопки, удерживая клавишу **Shift**, вызываем контекстное меню и выбираем команды **Center Horizontally** и потом **Center Vertically**. Кнопки равномерно распределятся по экрану.



Контекстное меню можно вызвать и напрямую у кнопок на экране. Достаточно указателем мыши начертить ограничивающий прямоугольник вокруг всех кнопок и щёлкнуть правой кнопкой. Но у разметки **ConstraintLayout** наблюдается неприятное поведение - после выравнивания кнопок последующий щелчок кнопки сдвигает их. Если у вас такого эффекта не происходит, то пользуйтесь.

В окне **Component Tree** выделите строку **button**. У вас должно появиться окно свойств **Properties**. Давайте избавимся от стандартных идентификаторов, а будем сразу приучаться давать осмысленные имена. Например, для первой кнопки присвоим свойству **id** значение **buttonRed** вместо стандартного **@+id/button**

Для второй кнопки присвоим значение **buttonYellow**.



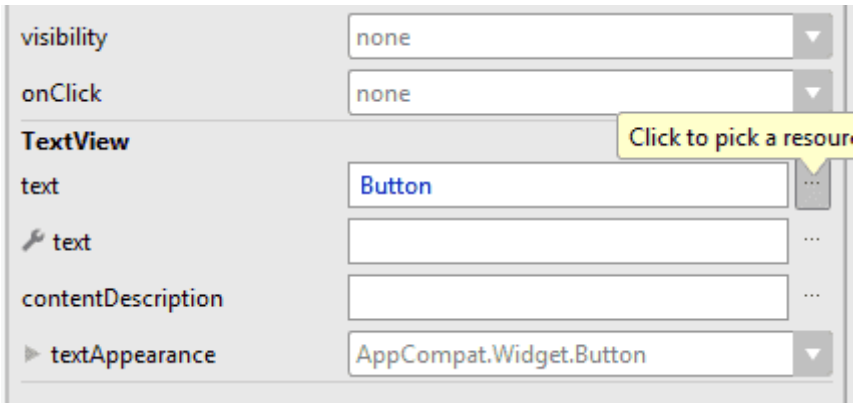
Аналогично настраиваем третью кнопку **buttonGreen**.

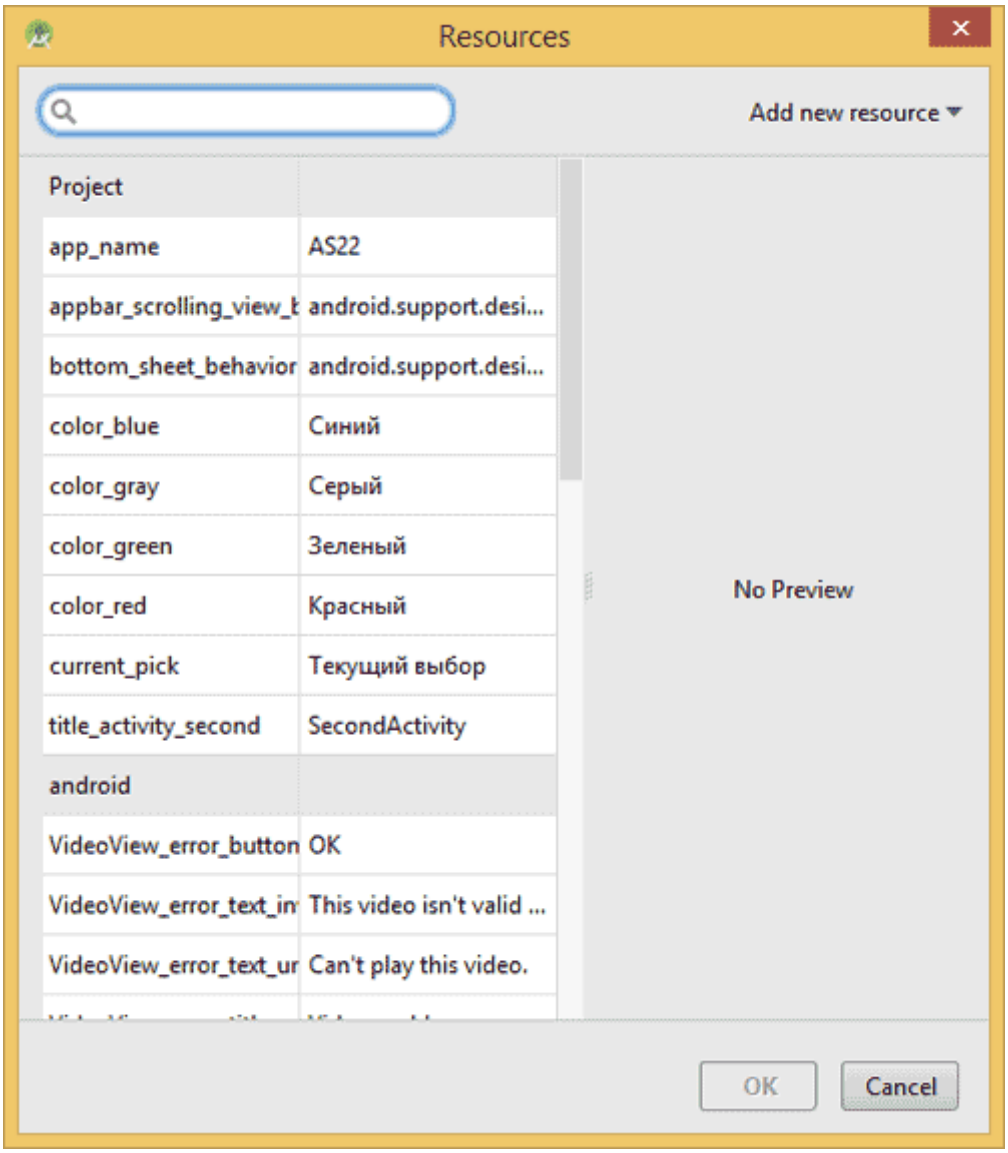
Строковые ресурсы

Теперь нам нужно заменить текст на кнопках на слова *Красный*, *Жёлтый* и *Зелёный*. На прошлом уроке мы просто присвоили свойству **Text** нужную строку. Но на самом деле это неправильный подход и даже среда разработки выводит предупреждающие значки у кнопок, если вы переключитесь в режим **Text**. По

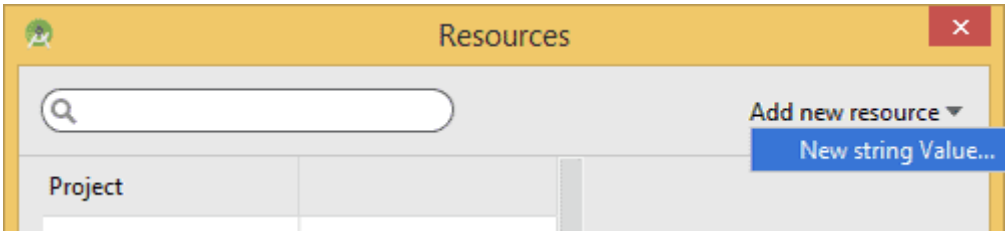
правилам, строки нужно хранить в строковых ресурсах. Подобный подход даёт разработчику множество преимуществ, в частности, быструю локализацию приложения. Считайте это стандартом, которого нужно придерживаться.

Процесс создания строковых ресурсов очень прост. Переключитесь обратно в режим **Design** и выберите кнопку *buttonRed*. В окне свойств найдите свойство **text**. Рядом находится кнопка с многоточием. Щёлкните на кнопке. У вас откроется диалоговое окно **Resources**.

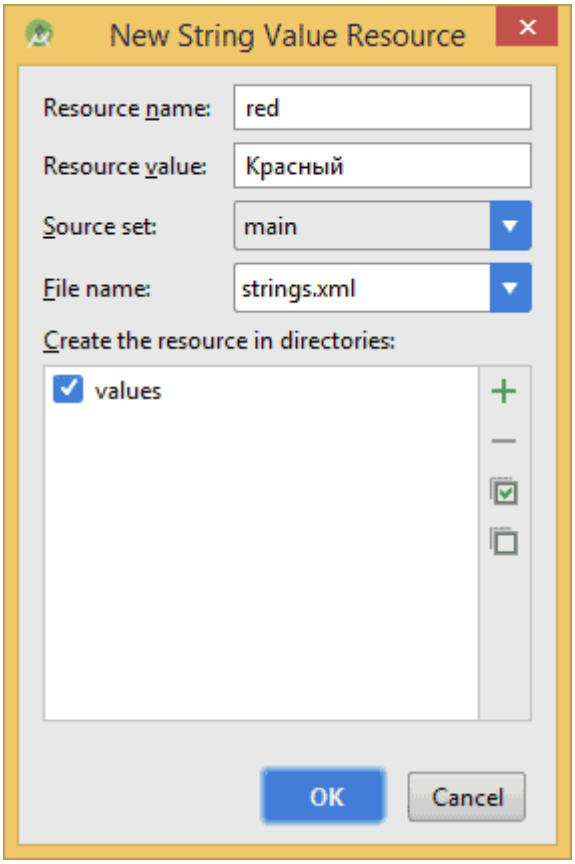




Нажмите на выпадающий список **Add new resource** для создания нового строкового ресурса и выберите **New String Value**.



В новом окне **New String Value Resource** в первом поле **Resource Name** введите название ресурса, например, **red**, а во втором поле **Resource Value** введите текст для кнопки (напр. Красный). Остальные поля не трогаем. Аналогичным образом поступите с другими двумя кнопками (Жёлтый и Зёленый).



Программно можно добиться такого же результата, отредактировав файл **strings.xml**, который находится в папке **res/values** вашего проекта. Сейчас он может выглядеть так.

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

    <string name="app_name">TrafficLight</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string name="red">Красный</string>
    <string name="yellow">Жёлтый</string>
    <string name="green">Зелёный</string>

</resources>
```

Мы совсем забыли про элемент **TextView**. Исправим упущение. Разместим компонент под кнопками и снова сделаем выравнивание.

Пусть на нём выводится текст, извещающий о текущем цвете фона приложения. Так как в ресурсах у нас уже есть слова *Красный, Жёлтый и Зелёный*, изначально предназначенные для кнопок, то мы не будем создавать новые строковые ресурсы, а воспользуемся готовыми наработками. По умолчанию у нас используется красный цвет. В окне свойств выбираем свойство **text** для **TextView** и нажимаем кнопку с многоточием для вызова знакомого диалогового окна. На этот раз мы не будем щёлкать на кнопке **New Resource**, а сразу выберем строку **red**, которая, как мы помним, содержит текст **Красный** и щёлкнем кнопку **OK** (можно сделать сразу двойной щелчок на строке).

Заодно расскажу о другой настройке под ней с изображением гаечного ключа и таким же названием **text**. Представьте себе, что в реальной программе изначально у **TextView** нет никакого текста, он будет сформирован позже программно. Но чтобы разработчик видел, как будет выглядеть дизайн экрана, ему нужно видеть текст на этапе проектирования. Для этих целей и предназначена вторая настройки для текста. Подобное вы можете увидеть и у других свойства компонента.

С текстом разобрались. Но где остальные настройки? В окне свойств **Properties** нажмите на ссылку **View all properties**. Тут увидите все доступные атрибуты для компонента. Если вы не помните точное название, то воспользуйтесь поиском. Я помню, что для размера шрифта используется что-то со словом **size**. Вспомнил, это **textSize**.


TextView

Выбираем нужное значение из выпадающего списка. Студия предлагает варианты, которые рекомендованы по правилам Material Design. Если вы хотите установить своё значение, то переключитесь в режим **Text** и вручную поменяйте значение. Если этого не сделать, а пытаться отредактировать значение в режиме **Design**, то студия не будет реагировать.

Рекомендую постоянно переключаться в режим **Text** и смотреть, что происходит в коде. Это позволит вам увереннее разбираться в коде и читать чужой код. Как правило, новички предпочитают работать через визуальные инструменты, а программисты с опытом самостоятельно пишут практически весь код. Нужно найти разумный баланс между двумя подходами. Всё придёт со временем.

Со строками вроде разобрались. Давайте теперь в ресурсах зададим цвета для фона программы. Ресурсы для цветов принято хранить в отдельном файле **res/values/colors.xml**, хотя технически никто не запрещает хранить их в том же файле **strings.xml**.

Откроем указанный файл и добавим ресурс жёлтого цвета между тегами **resources**:

```
<color name="yellowColor">#FFFF00</color>
```

Слева появится жёлтый квадрат, по нему легко видеть цвет заданного ресурса.

По такому же принципу добавьте зелёный цвет.

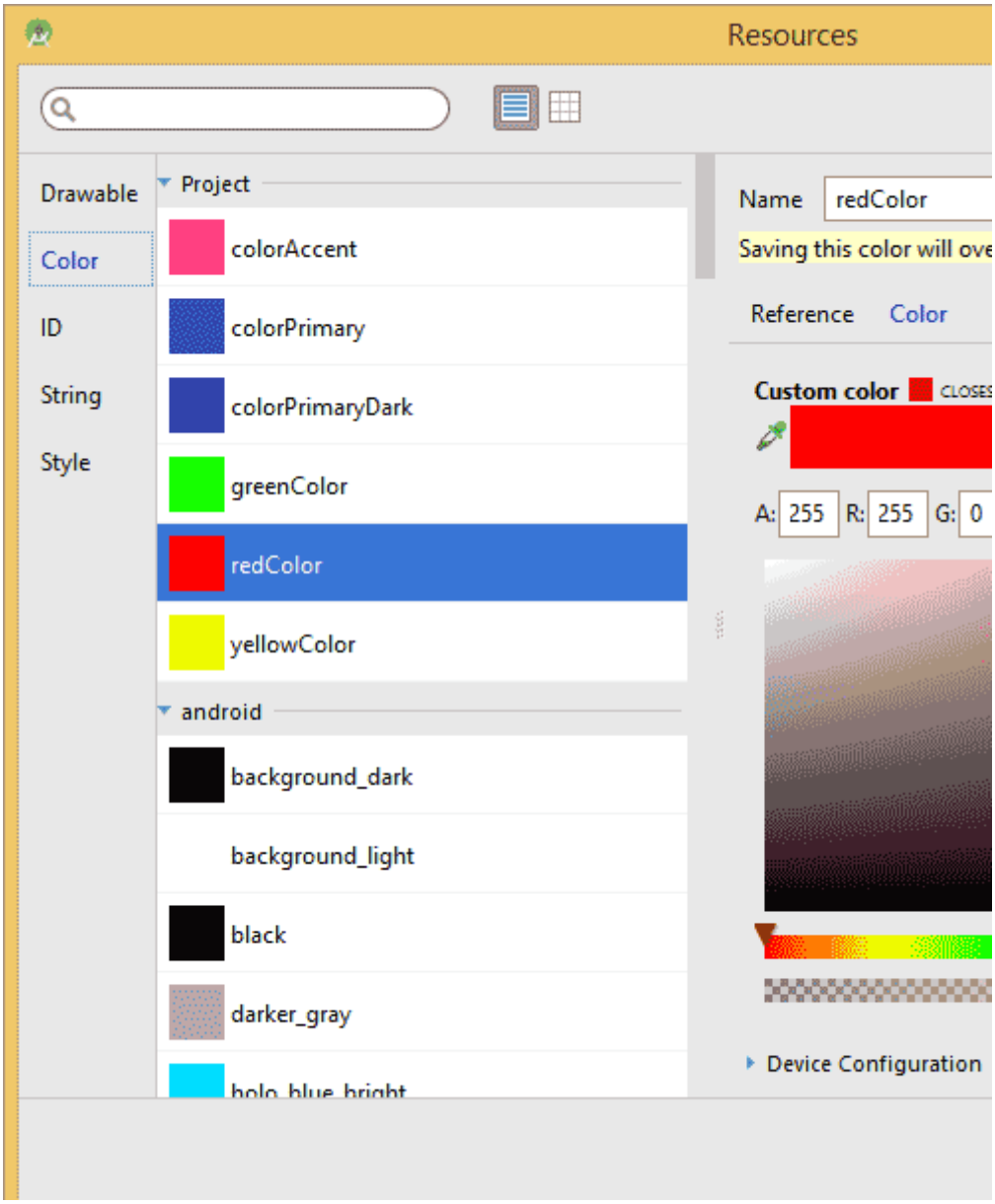
```
<color name="greenColor">#FF00FF00</color>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>

    <color name="yellowColor">#FFFF00</color>
    <color name="greenColor">#FF00FF00</color>

</resources>
```

Самостоятельно добавьте ресурс для красного цвета под именем **redColor**.

Если вам знаком такой формат цвета, то трудностей добавление новых цветов подобным способом вам не доставит. Если нужно выбрать более сложный цвет, то проще воспользоваться мастером, как мы это делали в уроке "Hello Kitty", когда выбирали розовый цвет, а затем полученный цвет скопировать в ресурсы.

Определив в ресурсах все необходимые цвета, можно сразу присвоить красный цвет для контейнера **ConstraintLayout**. В окне свойств находим для данного элемента свойство **background** (чтобы увидеть все свойства компонента, нажмите ссылку **View all properties**). Снова нажимаем кнопку с тремя точками, чтобы открыть диалоговое окно. В окне выбираем раздел **Color** и ищем свой ресурс **redColor**.



Если посмотреть в текстовом режиме, то увидите строчку **android:background="@color/redColor"** для тега **ConstraintLayout**.

Так как мы будем менять фон у **ConstraintLayout**, то присвоим ему идентификатор.

Общий каркас приложения завершен. У нас есть три кнопки с соответствующими текстами, текстовая надпись со словом *Красный*, и красный фон, используемый в контейнере **ConstraintLayout**. Пора приступать к программной логике программы. А пока можно запустить приложение, чтобы убедиться, что мы не сделали ошибок в разметке.

Код для программы

Наша задача - обработать щелчки трёх кнопок и менять цвет фона приложения, а также текст в **TextView**. На прошлом занятии мы уже познакомились с удобным способом обработки события **onClick**. Давайте закрепим пройденный материал и повторим тот же код для первой кнопки. Пропишем вручную событие **onClick** в теге **Button**:

```
android:onClick="onRedButtonClick"
```

Вспоминаем - в режиме **Text** помещаем курсор на названии метода и нажимаем комбинацию **Alt+Enter**, чтобы создать заготовку щелчка первой кнопки в классе **MainActivity**.

Объявим переменные в классе и получим к ним доступ в методе **onCreate()**:

```
// до метода onCreate()
private ConstraintLayout mConstraintLayout;
private TextView mInfoTextView;

// в методе onCreate()
mConstraintLayout = (ConstraintLayout) findViewById(R.id.constraintLayout);
mInfoTextView = (TextView) findViewById(R.id.textViewInfo);
```

Напоминаю, что код нужно писать вручную, а не копировать с сайта. Тогда многие вопросы отпадут и вам не придётся бежать на форум с криками, что ничего не работает.

Кстати, этот код вызвал трудности у новичков. Меня завалили письмами и устроили вой на форуме. Раньше в студии по умолчанию использовался компонент **RelativeLayout** и пример был написан под него. Новички размещали кнопки в **ConstraintLayout**, а в код вставляли строчки:

```
private RelativeLayout mRelativeLayout;

mRelativeLayout = (RelativeLayout)findViewById(R.id.relativeLayout);
```

Для не слишком догадливых программистов подчёркиваю - **ConstraintLayout** и **RelativeLayout** - это разные вещи. Вы же не путаете кота с собакой, хотя у них есть и хвосты и лапы. Не нужно писать код с использованием несуществующих компонентов, точнее не надо копировать и вставлять мой код. Иначе никогда не научитесь писать программы.

Пишем код для щелчка кнопки с надписью "Красный":

```
public void onRedButtonClick(View view) {
    mInfoTextView.setText(R.string.red);
    mConstraintLayout.setBackgroundColor(ContextCompat.getColor(this, R.color.redColor));
}
```

Мы обращаемся к созданным ресурсам через специальный класс **R** и через точку указываем тип ресурсов, а затем имя ресурса.

Для кнопки "Зелёный" напишите код самостоятельно, добавив метод **onGreenButtonClick()**.

Для кнопки "Жёлтый" напишем код в традиционной манере через слушателя **OnClickListener**. В методе **onCreate()** добавляем:

```
Button yellowButton = (Button) findViewById(R.id.buttonYellow);
yellowButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        mInfoTextView.setText(R.string.yellow);
        mConstraintLayout.setBackgroundColor(ContextCompat
                .getColor(MainActivity.this, R.color.yellowColor));
    }
});
```

Раньше для изменения кода использовался код:

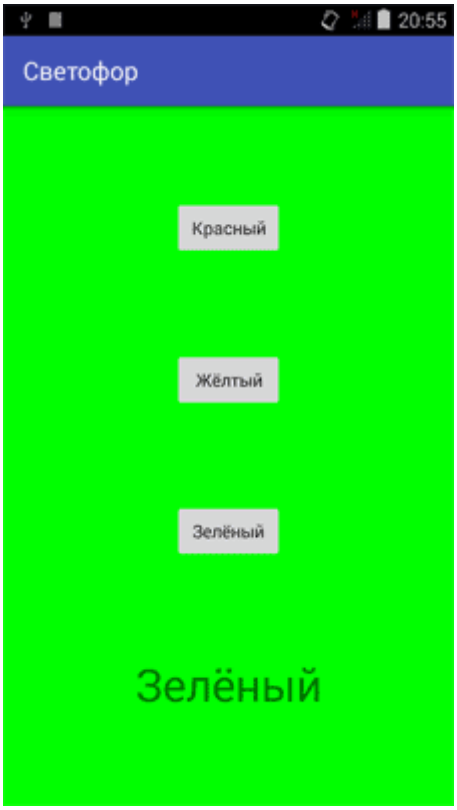
```
mRelativeLayout.setBackgroundColor(getResources().getColor(R.color.redColor));
```

В Android 6 (API 23) метод **getColor(int id)** объявили устаревшим и студия теперь подчёркивает данный метод. Можно заменить на один из двух вариантов:

```
// у второго параметра используем значение null
mRelativeLayout.setBackgroundColor(getResources().getColor(R.color.yellowColor, null));

// используем метод из библиотеки совместимости
mRelativeLayout.setBackgroundColor(ContextCompat.getColor(MainActivity.this, R.color.yellowCo
lor));
```

Первый вариант подойдёт, если вы не поддерживаете старые устройства, а сразу пишете приложение для телефонов с API 23. Второй способ более универсальный. Его я и использовал.

Запускаем приложение и щёлкаем по кнопкам - текст в надписи и фон в приложении должны меняться в соответствии с нажатой кнопкой.



Полный текст кода будет выглядеть следующим образом:

```
// Если этот код работает, его написал Александр Климов,
// а если нет, то не знаю, кто его писал.
package ru.alexanderklimov.trafficlight;

import android.os.Bundle;
import android.support.constraint.ConstraintLayout;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    private ConstraintLayout mConstraintLayout;
    private TextView mInfoTextView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mConstraintLayout = (ConstraintLayout) findViewById(R.id.constraintLayout);
        mInfoTextView = (TextView) findViewById(R.id.textViewInfo);

        Button yellowButton = (Button) findViewById(R.id.buttonYellow);
        yellowButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mInfoTextView.setText(R.string.yellow);
                mConstraintLayout.setBackgroundColor(ContextCompat
                        .getColor(MainActivity.this, R.color.yellowColor));
            }
        });
    }

    public void onRedButtonClick(View view) {
        mInfoTextView.setText(R.string.red);
        mConstraintLayout.setBackgroundColor(ContextCompat.getColor(this, R.color.redColor));
    }

    public void onGreenButtonClick(View view) {
        mInfoTextView.setText(R.string.green);
        mConstraintLayout.setBackgroundColor(ContextCompat.getColor(this, R.color.greenColo
r));
    }
}
```

В качестве домашнего задания упростите код для трёх кнопок, создав для них общий метод **onClick()** (почитайте статью про кнопки (http://developer.alexanderklimov.ru/android/views/button.php)).

Приведённая ниже информация немного устарела. В Android 5.0 используется другой стиль оформления заголовка программы и значка там теперь нет (хотя его можно туда поместить). Но значки всё равно нужны для отображения вашей программы на домашнем экране или в лаунчерах.

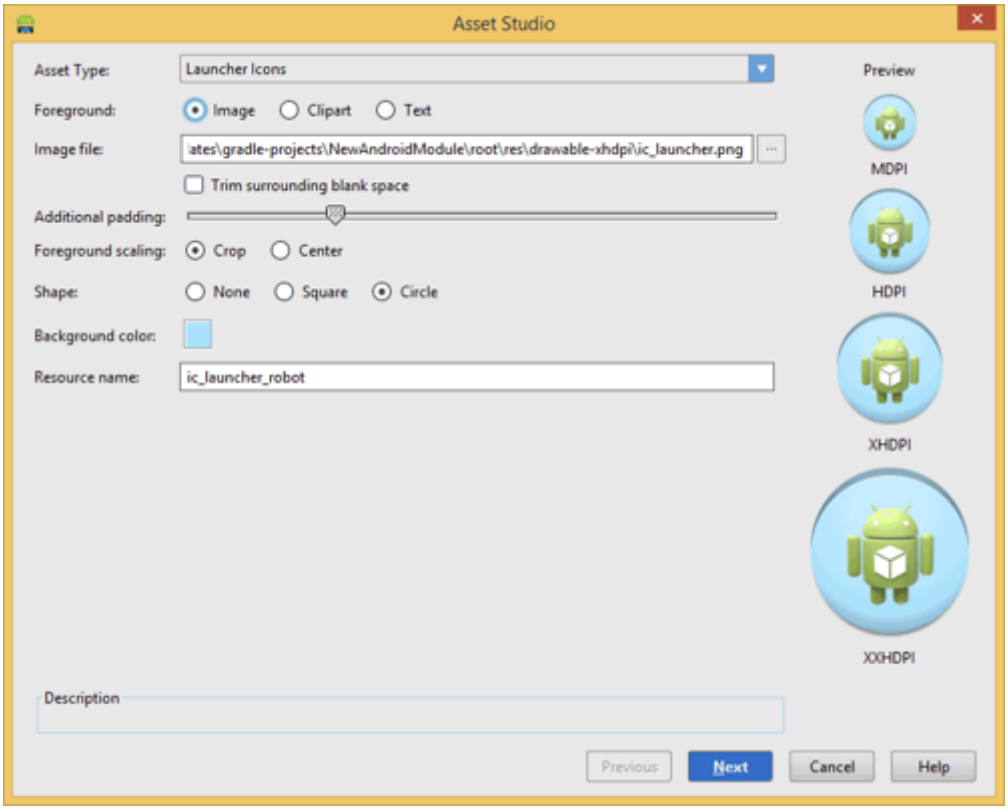
Поговорим о значках. По умолчанию студия использует изображение зелёного робота в качестве значка для вашей программы. Откройте в студии папку **res/mipmap**. Я уже рассказывал, что эта папка является виртуальной и в действительности существуют папки **res/mipmap-hdpi**, **res/mipmap-mdpi**, **res/mipmap-xhdpi**, **res/mipmap-xxhdpi**. В каждой из этих папок есть файл с одинаковым именем **ic_launcher.png** и недавно добавили ещё **ic_launcher_round.png** для Android 7. Вся разница между этими файлами заключается в размерах. В зависимости от разрешения экрана на устройстве система выбирает наиболее подходящий по размеру изображение и выводит его в качестве значка в заголовке приложения и на домашнем экране. Самый простой вариант заменить стандартное изображение на своё - создать своё изображение и заменить им имеющийся значок. Рекомендуется создавать под каждое разрешение свой значок. Причем здесь указаны не все варианты. В таком случае вам нужно создать самостоятельно папку, например, **mipmap-xxxhdpi** и разместить там картинку требуемого размера. Если вы пропустите какой-то размер, то система попробует взять какой-нибудь значок с этим именем из другой папки и смаштабировать его. Но лучше так не делать.

Если вы не хотите менять существующие стандартные значки, а хотите использовать значки под другим именем, то в этом случае подготовьте все необходимые размеры, разместите их во всех папках **mipmap-** под своим именем, а затем в манифесте замените строчку у атрибута **android:icon**:

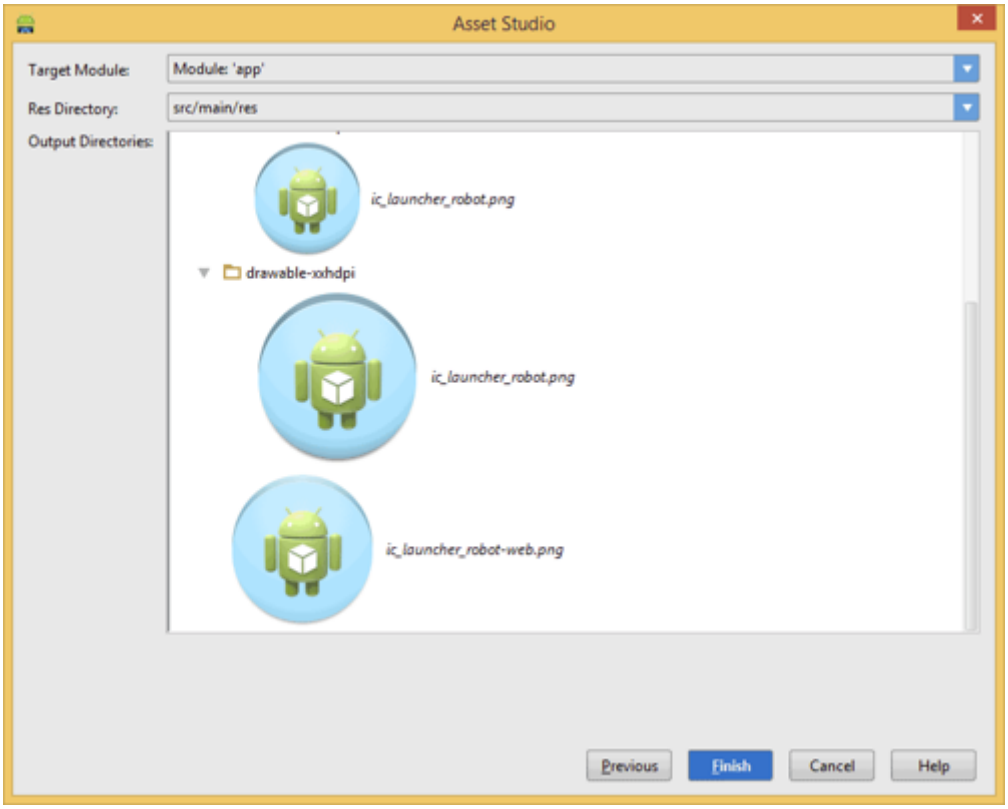
```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher_cat"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
```

В состав студии входит набор предопределённых значков и генератор собственных значков. Щёлкните правой кнопкой на папке **drawable** или **mipmap** и выберите в меню **New | Image Asset**.

Откроется диалоговое окно, где вы можете указать в качестве источника файл на компьютере, вариант из клипарта или набор символов. Также вы можете задать форму значка, цвет фона и прочие параметры. Уверен, что вы разберётесь самостоятельно.



(http://fotki.yandex.ru/users/netsources/view/614526)



(http://fotki.yandex.ru/users/netsources/view/614527)

Кроме значков для различных разрешений, генератор создаст дополнительный файл с суффиксом **-web**, который будет скопирован в папку **main**. Этот файл используется для Google Play, когда будете размещать приложение в магазине приложений и давать описание к нему.

Дополнительное чтение