

Виды Layouts. Ключевые отличия и свойства.

Расположение View-элементов на экране зависит от **ViewGroup** (Layout), в которой они находятся. Рассмотрим основные виды **Layout**.

LINEARLAYOUT (LL)

LinearLayout – отображает View-элементы в виде одной строки (если он Horizontal) или одного столбца (если он Vertical).

Этот вид ViewGroup по умолчанию предлагается при создании новых layout-файлов. Он действительно удобен и достаточно гибок, чтобы создавать экраны различной сложности. LL имеет свойство Orientation, которое определяет, как будут расположены дочерние элементы – горизонтальной или вертикальной линией.

Рассмотрим пример. Создадим проект:

Project name: P0061_Layouts

Build Target: Android 2.3.3

Application name: Layouts

Package name: ru.startandroid.develop.layouts

Create Activity: MainActivity

Откроем layout-файл **main.xml**, и поместим в него следующий код:

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

<LinearLayout

    xmlns:android="http://schemas.android.com/apk/res/android"

    android:layout_width="match_parent"

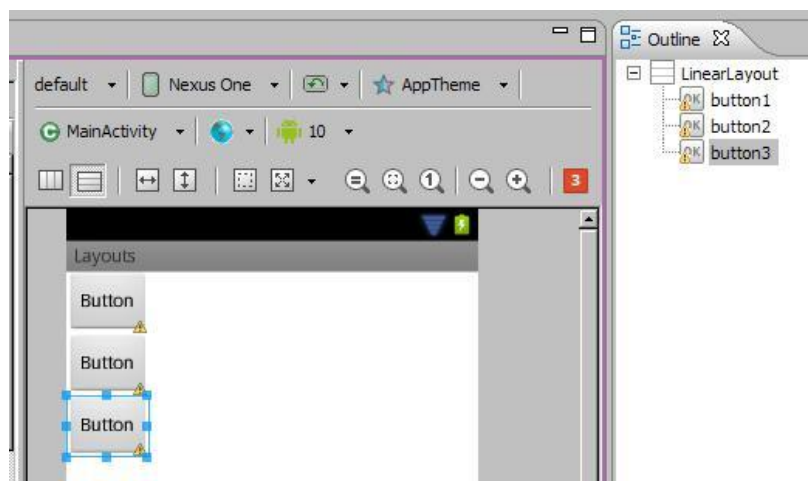
    android:layout_height="match_parent"

    android:orientation="vertical">

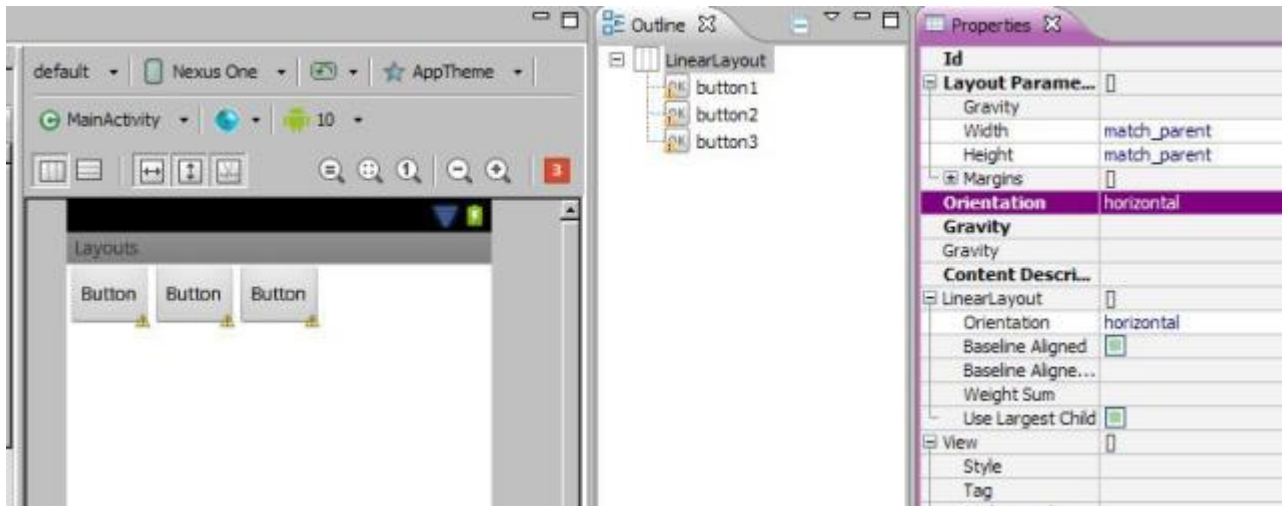
</LinearLayout>
```

Теперь корневой элемент у нас LinearLayout с вертикальной ориентацией.

Перетащите слева в корневой LinearLayout три кнопки. Они выстроились вертикально:

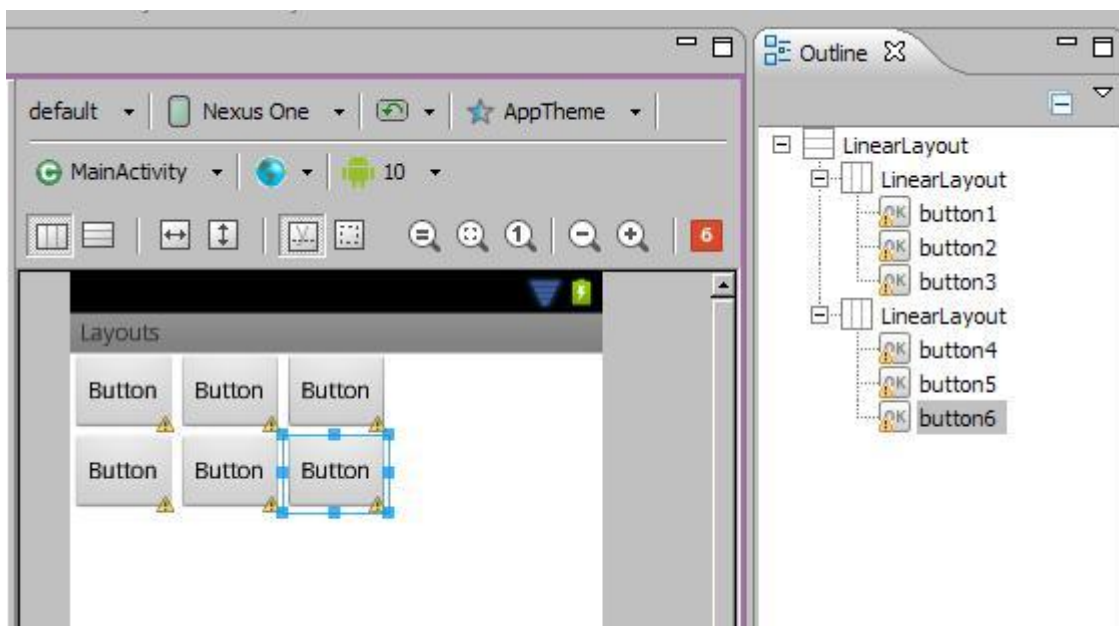


Теперь в Properties меняем для LL свойство **Orientation** на **horizontal** и сохраняем (CTRL+SHIFT+S) – кнопки выстроились горизонтально.



GroupView можно вкладывать друг в друга. Вложим в один LL два других. Удалите в main.xml все элементы (три кнопки) кроме корневого LL. Ориентацию корневого LL укажем вертикальную и добавим в него два новых горизонтальных LL. В списке элементов слева они находятся в разделе Layouts.

В каждый горизонтальный LL добавим по три кнопки. Получилось два горизонтальных ряда кнопок:



TABLELAYOUT (TL)

TableLayout – отображает элементы в виде таблицы, по строкам и столбцам.

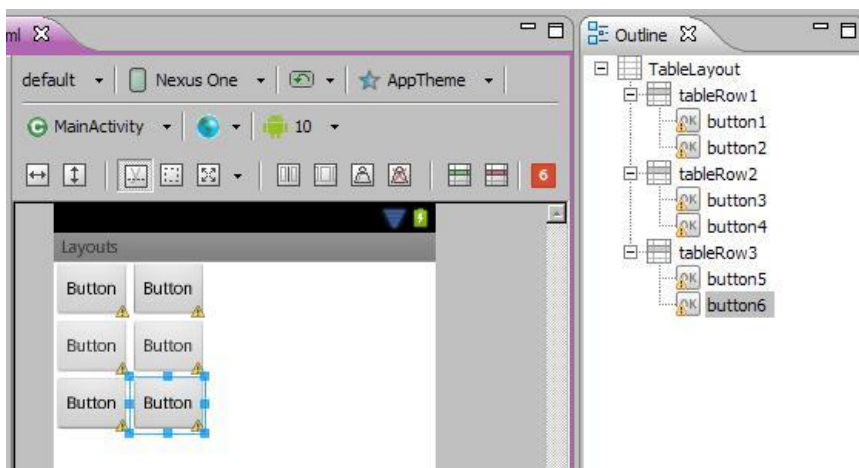
TL состоит из строк **TableRow** (TR). Каждая TR в свою очередь содержит View-элементы, формирующие столбцы. Т.е. кол-во View в TR - это кол-во столбцов. Но кол-во столбцов в таблице должно быть равным для всех строк. Поэтому, если в разных TR - разное кол-во View-элементов (столбцов), то общее кол-во определяется по TR с максимальным кол-вом.

Рассмотрим на примере.

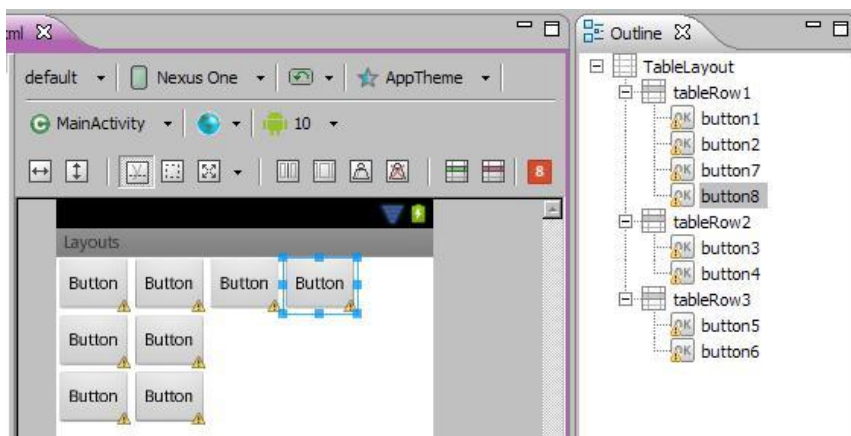
Создадим layout-файл **tlayout.xml**. с корневым элементом **TableLayout**:



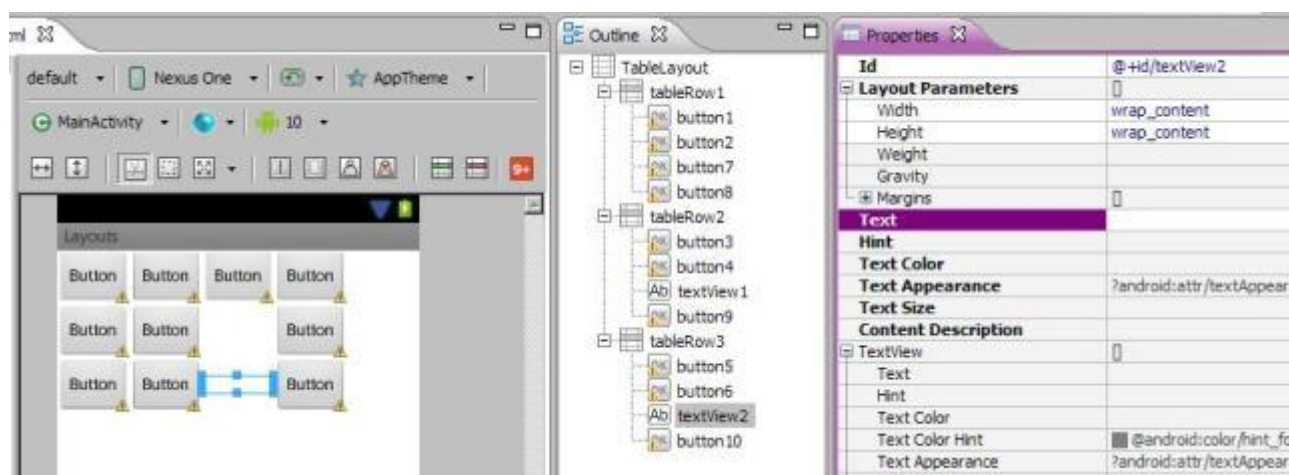
Добавим в корневой **TableLayout** три **TableRow**-строки (из раздела **Layouts** слева) и в каждую строку добавим по две кнопки. Результат: наша таблица имеет три строки и два столбца.



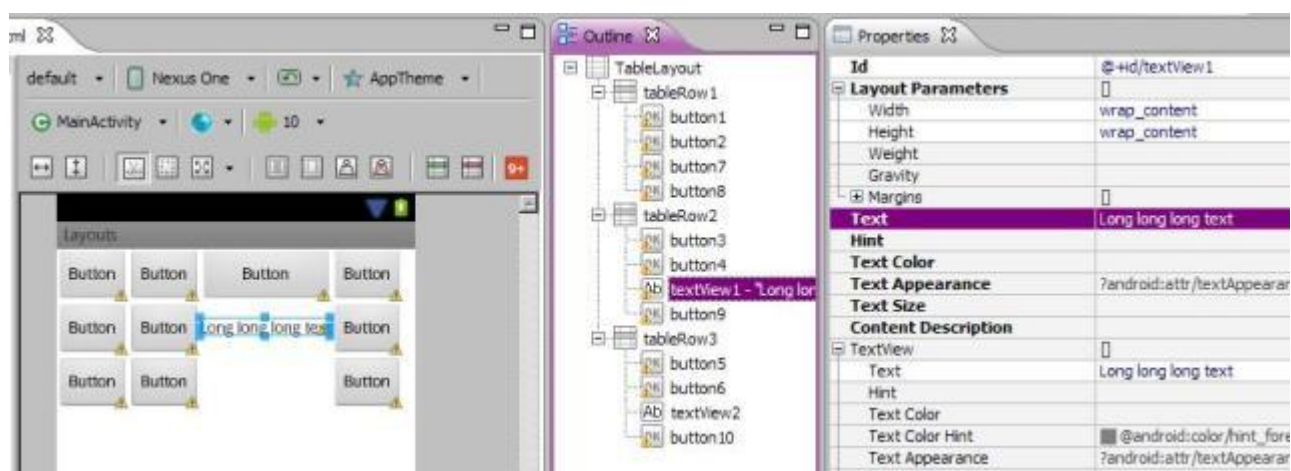
Добавим в первую строку еще пару кнопок. Кол-во столбцов для всех строк теперь равно 4, т.к. оно определяется по строке с максимальным кол-вом элементов, т.е. по первой строке. Для второй и третьей строки третий и четвертый столбцы просто ничем не заполнены.



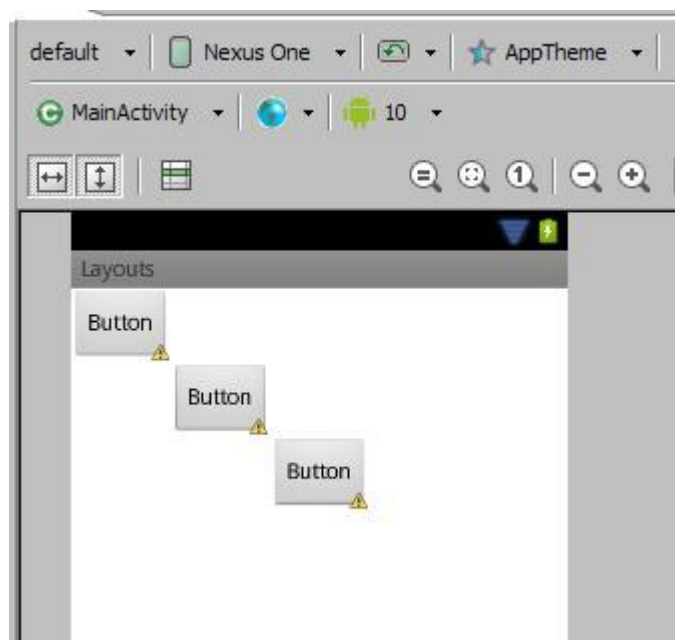
Во вторую строку добавим TextView и Button, и текст в добавленном TextView сделаем пустым. В третьей строке сделаем то же самое. Мы видим, что эти элементы легли в третий и четвертый столбец. И т.к. TextView у нас без текста и на экране не виден, кажется что третий столбец во второй и третьей строке пустой.



Ширина столбца определяется по самому широкому элементу из этого столбца. Введем текст в один из TextView и видим, что он расширил столбец.



Попробуйте сами сделать такое упражнение:



RELATIVELAYOUT (RL)

RelativeLayout – для каждого элемента настраивается его положение относительно других элементов.

В этом виде Layout каждый View-элемент может быть расположен определенным образом относительно указанного View-элемента.

Виды отношений:

- 1) слева, справа, сверху, снизу указанного элемента (layout_toLeftOf, layout_toRightOf, layout_above, layout_below)
- 2) выравненным по левому, правому, верхнему, нижнему краю указанного элемента (layout_alignLeft, layout_alignRight, layout_alignTop, layout_alignBottom)
- 3) выравненным по левому, правому, верхнему, нижнему краю родителя (layout_alignParentLeft, layout_alignParentRight, layout_alignParentTop, layout_alignParentBottom)
- 4) выравненным по центру вертикально, по центру горизонтально, по центру вертикально и горизонтально относительно родителя (layout_centerVertical, layout_centerHorizontal, layout_centerInParent)

Подробнее можно почитать в [Документации](#).

Создадим **layout.xml** и скопируем туда такой xml-код:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/label"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Type here:">
    </TextView>
    <EditText
        android:id="@+id/entry"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@+id/label"
        android:background="@android:drawable/editbox_background">
```

```

</EditText>

<Button
    android:id="@+id/ok"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/entry"
    android:layout_marginLeft="10dip"
    android:text="OK">

</Button>

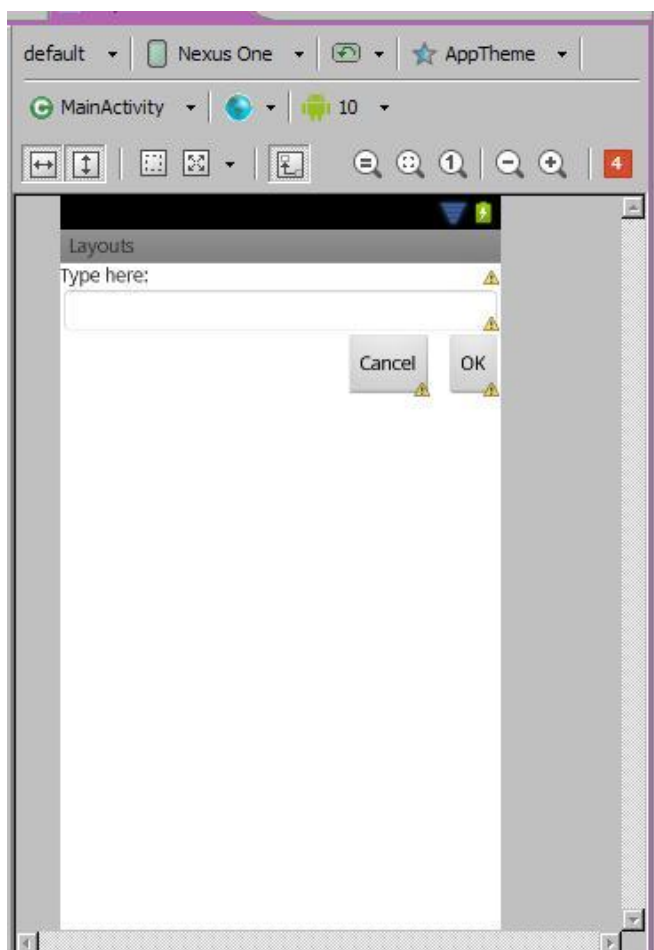
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/ok"
    android:layout_toLeftOf="@+id/ok"
    android:text="Cancel">

</Button>

</RelativeLayout>

```

Здесь у нас корневой элемент - RelativeLayout. Получился такой экран:



Рассмотрим незнакомые атрибуты и их значения:

```
android:layout_width="match_parent"  
android:layout_height="wrap_content"  
android:id="@+id/entry"
```

- слово **android** в названии каждого атрибута – это **namespace**.

- **id** – это ID элемента,

- **layout_width** (ширина элемента) и **layout_height** (высота элемента) могут задаваться в абсолютных значениях, а могут быть следующими: **fill_parent** (максимально возможная ширина или высота в пределах родителя) и **wrap_content** (ширина или высота определяется по содержимому элемента). В [Документации](#) указывается, что есть еще **match_parent**. Это то же самое, что и **fill_parent**. По каким-то причинам, разработчики системы решили, что название **match_parent** удобнее, и от **fill_parent** постепенно будут отказываться. А пока его оставили для совместимости. Так что запомните, что **match_parent** = **fill_parent** и в дальнейшем будем стараться использовать **match_parent**.

В данном примере используются TextView, EditText и два Button – OK и Cancel. Рассмотрим их атрибуты:

TextView

```
android:id="@+id/label" - ID  
android:layout_width="match_parent" - занимает всю доступную ему ширину (хоть это и не видно на  
экране);  
android:layout_height="wrap_content" - высота по содержимому;  
ни к чему никак не относится
```

EditText

```
android:id="@+id/entry" - ID  
android:layout_width="match_parent" - вся доступная ему ширина  
android:layout_height="wrap_content" - высота по содержимому  
android:layout_below="@+id/label" - расположен ниже TextView (ссылка по ID)
```

Button_OK

```
android:id="@+id/ok" – ID  
android:layout_width="wrap_content" - ширина по содержимому  
android:layout_height="wrap_content" – высота по содержимому  
android:layout_below="@+id/entry" - расположен ниже EditText  
android:layout_alignParentRight="true" - выравнен по правому краю родителя  
android:layout_marginLeft="10dip" – имеет отступ слева (чтобы Button_Cancel был не впритык)
```

Button_Cancel

```
android:layout_width="wrap_content" - ширина по содержимому  
android:layout_height="wrap_content" – высота по содержимому  
android:layout_toLeftOf="@+id/ok" - расположен слева от Button_OK  
android:layout_alignTop="@+id/ok" - выравнен по верхнему краю Button_OK
```

Вы можете добавлять элементы и экспериментировать с их размещением...

Обратите внимание, что у View-элемента может не быть ID (android:id). Например, для TextView он обычно не нужен, т.к. они чаще всего статичны и мы к ним почти не обращаемся при работе приложения. Другое дело EditText – мы работаем с содержимым текстового поля, и Button – нам надо обрабатывать нажатия и

соответственно знать, какая именно кнопка нажата. В будущем мы увидим еще одну необходимость задания ID для View-элемента.

ABSOLUTELAYOUT (AL)

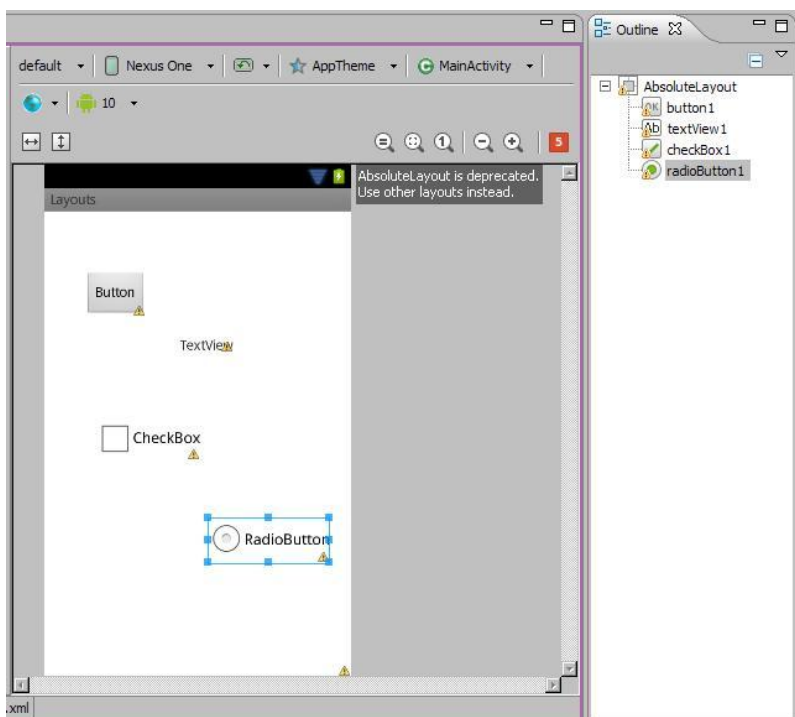
AbsoluteLayout — для каждого элемента указывается явная позиция на экране в системе координат (x,y)

Обеспечивает абсолютное позиционирование элементов на экране. Вы указываете координаты для левого верхнего угла компонента.

Создадим **layout.xml** с корневым **AbsoluteLayout**



Теперь попробуйте перетаскиванием добавлять различные элементы на экран. Они не выстраиваются, как при **LinearLayout** или **TableLayout**, а ложатся там, куда вы их перетаскили. Т.е. это **абсолютное позиционирование**:



При открытии соответствующего xml-кода можно увидеть, что для задания координат используются **layout_x** и **layout_y**.

Вначале кажется, что это наиболее удобный и интуитивно понятный способ расположения элементов на экране - они сразу располагаются там где надо. Но это только в случае, когда вы разрабатываете для экрана с конкретным разрешением. Если открыть такое приложение на другом экране, все элементы сместятся и получится не так, как вы планировали. Поэтому этот Layout не рекомендуется использовать. И его совместимость с будущими версиями Android не гарантируется.

Есть еще [много видов](#) ViewGroup... Будем рассматривать их по мере необходимости.