Мы [говорили о ConstraintLayout](https://www.bignerdranch.com/blog/constraintlayout-vs-auto-layout-how-do-they-compare/) в предыдущем посте и сравнил его с авто макет. С тех пор прошел путь от Альфа-до бета-версии, и последнюю версию (beta4) пакеты дальнейшего быстрого улучшения и исправления. Кроме того, Студия Android 2.3 (в настоящее время канарейки 2, еще не финал) внесены улучшения в редактор.

В этой статье, мы рассмотрим несколько примеров.

Пример: Центрирование Видом

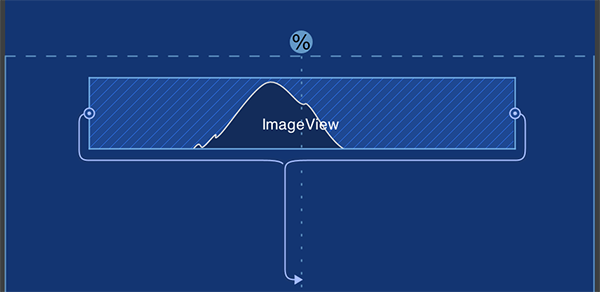
Автоматический макет для iOS и MacOS выставляет centerX и centerY якоря. Центрирование в его родительский вид так же просто, как добавление ограничения между представлением и якоря родительского центра. Аналогично, по центру-выравнивание нескольких видов достигается путем создания ограничений между их центрами.

ConstraintLayout не имеет якорей центра, но есть несколько механизмов, которые могут достичь того же эффекта.

Первый подход заключается в создании руководствоваться процентное ограничение установлено до 0,5: app:layout\_constraintGuide\_percent="0.5", и для согласования представления левого и правого якоря по направляющей:

app:layout\_constraintLeft\_toLeftOf="@+id/guideline"

app:layout\_constraintRight\_toLeftOf="@+id/guideline"



Второй подход состоит в том, чтобы ограничить левого и правого якоря считает его родителя:

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintRight\_toRightOf="parent"



Цепи

ConstraintLayout Альфа 9 добавили функцию цепи, что позволяет достичь многих трюков, которые ранее потребовали бы вложенным макетом.

“Цепочка” - это набор представлений, которые соединены попарно посредством двунаправленного ограничения. Другими словами, простейшая цепь состоит из двух окон, которые имеют ограничения в отношении друг друга в одной плоскости, например:

<Button

...

app:layout\_constraintRight\_toLeftOf="@id/button4"

app:layout\_constraintHorizontal\_chainStyle="packed"

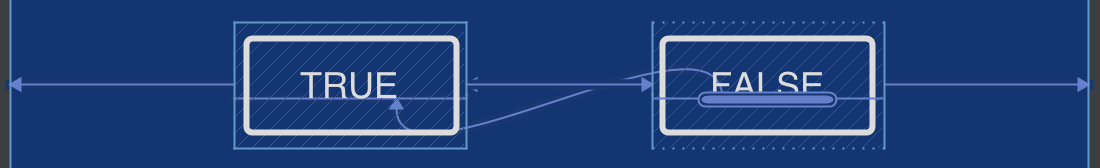
.../>

<Button

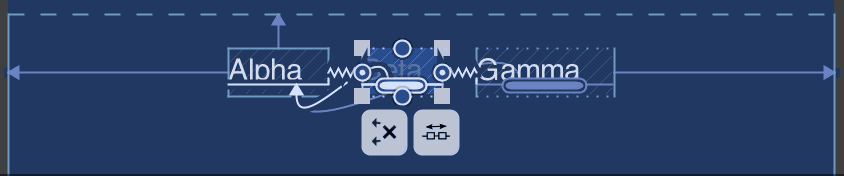
...

app:layout\_constraintLeft\_toRightOf="@+id/button3"

.../>

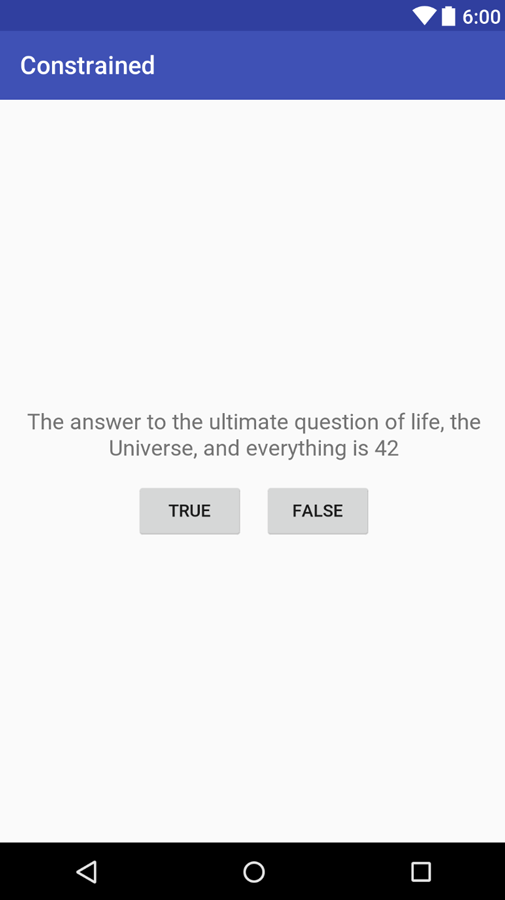


Цепи могут быть “распространение”, “упакованные” или “внутри”; это контролируется app:layout\_constraintHorizontal\_chainStyleсобственность (существует также Vertical версия). Делая упакованная цепь толкает взгляды близки друг к другу и делая его распространение выделяет пустое пространство вокруг них.



Пример: GeoQuiz

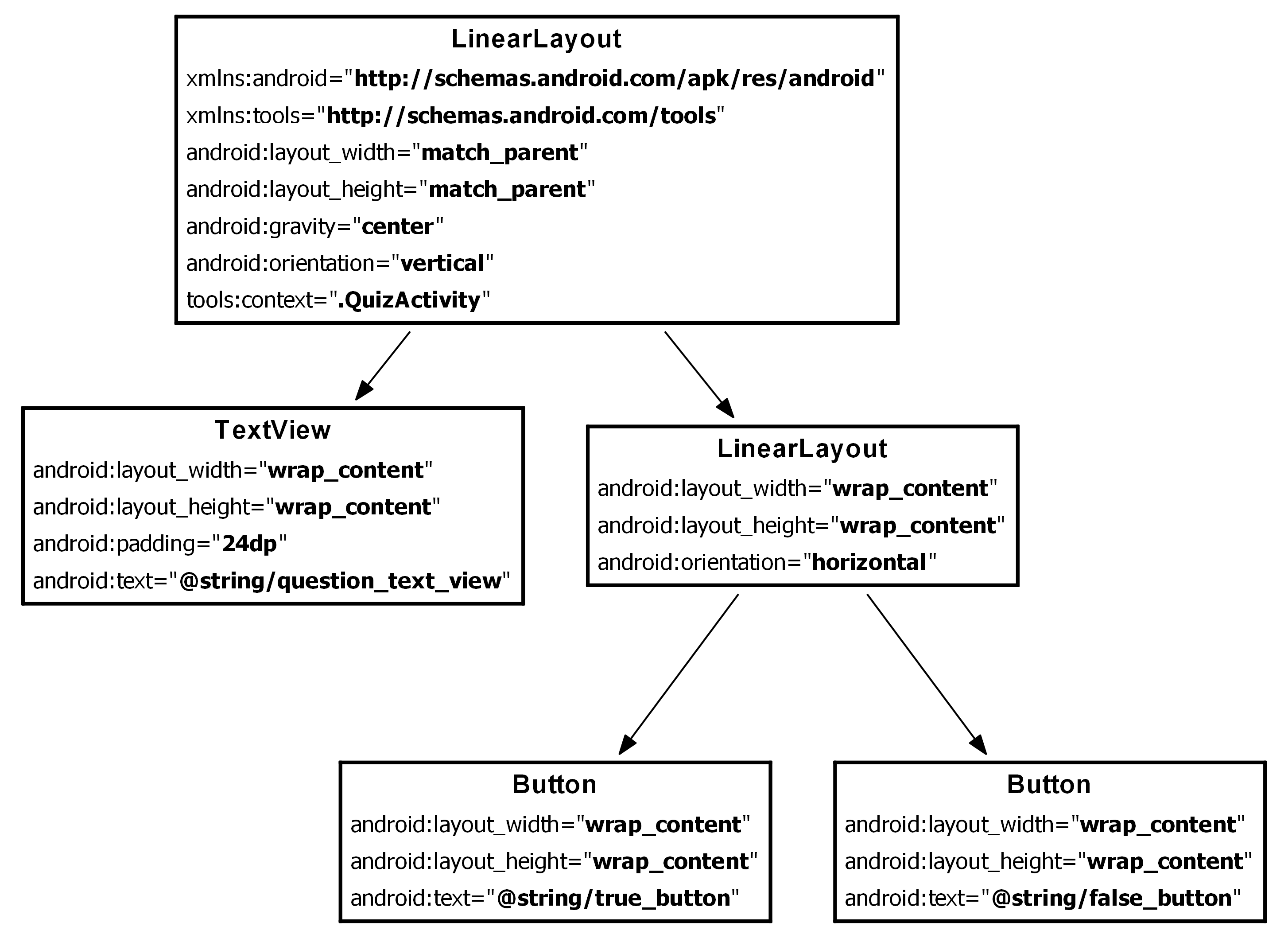
Первое упражнение в нашем [андроид руководство по программированию](https://www.bignerdranch.com/we-write/android-programming/)строит викторины приложение с пользовательским интерфейсом, который выглядит так:



Спецификации пользовательского интерфейса может сводиться к чему-то вроде этого:

* Он должен отображать просмотреть текст вопроса, с полями слева и справа, на 8 пунктов выше горизонтальной линии центра.
* Он должен отображать две кнопки, каждая из 8 точек по обе стороны от вертикальной линии центра, 8 пункта ниже горизонтальной линии центра.

Это макет, используемый в книге:

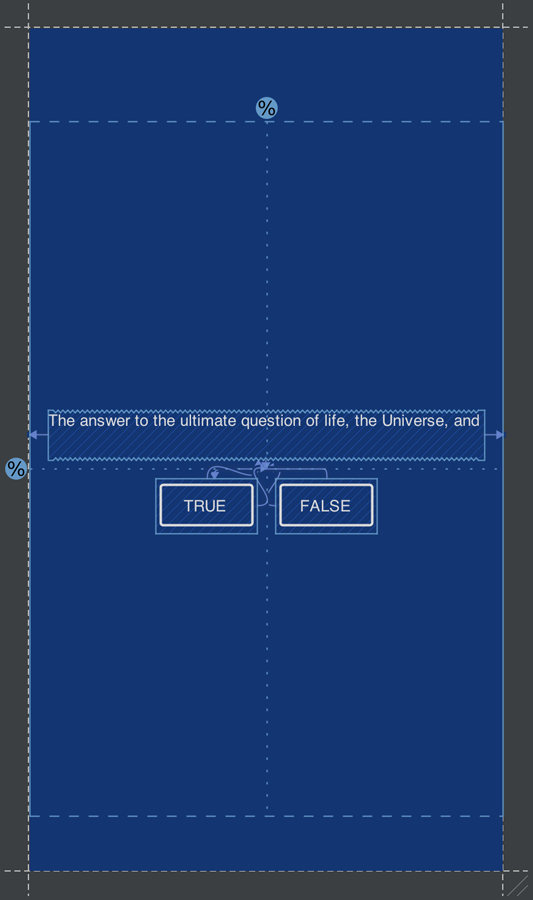


Для того, чтобы показать кнопки бок о бок ниже текст, в книге используются два вложенных LinearLayoutов. Это работает прекрасно, но в более сложных УИС карт раскроя может привести к проблемам производительности, достигнутые в результате некоторые детали реализации процесса компоновки. Когда вид или макет является недействительным (т. е. изменения содержания, размера или положения), его упаковку, или родительский вид, является также недействительным. Поскольку этот процесс является рекурсивным, единственное изменение в иерархии может привести к массе обновления. Таким образом, в целом, желательно иметь представление иерархии как можно более плоскими.

Уровень мельче позволяет уплощение иерархии представления, но, как упоминалось ранее, его недостатки делают его непригодным для использования в сложных УИС.

Новый малыш на блоке, ConstraintLayoutсделает эффективным макеты на основе ограничений на андроид. Что делает ConstraintLayout лучше RelativeLayout? Использование алгоритма Казуар и ограничений (каламбур) как ограничения, может быть установлен.

Вот тот же пользовательский интерфейс, разработанный с использованием ConstraintLayout ([макет XML-файл](https://www.bignerdranch.com/assets/img/blog/2016/12/example_01.xml)):

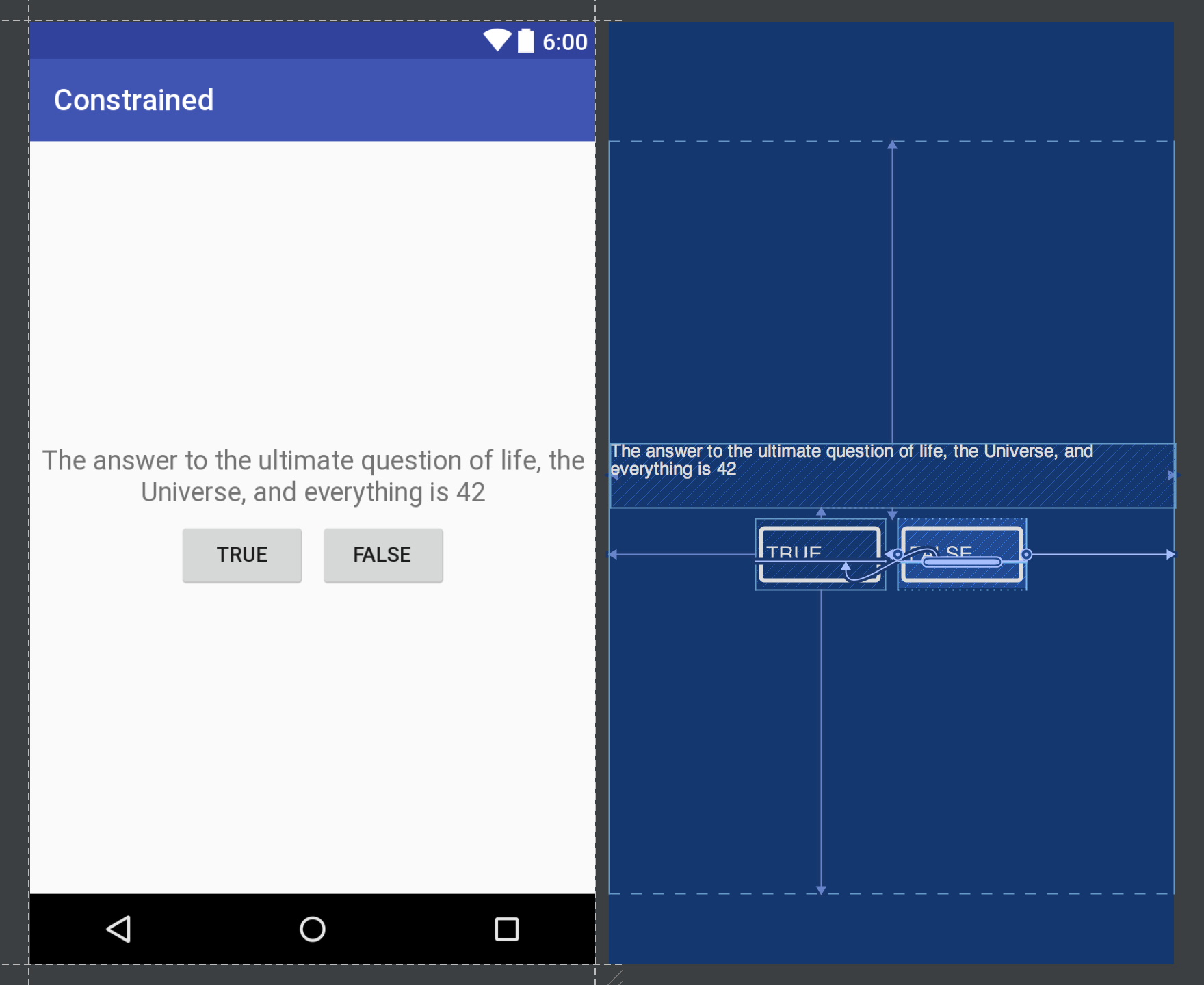


Это немного трудно, чтобы увидеть все ограничения, поэтому здесь они разбиты по частям:

* Пунктирные линии являются вертикальные и горизонтальные направляющие центра.
* Представление текста и две кнопки определить свое положение относительно этих принципов.

Этот макет также может быть создан без использования руководящих принципов. Цепи к спасению!

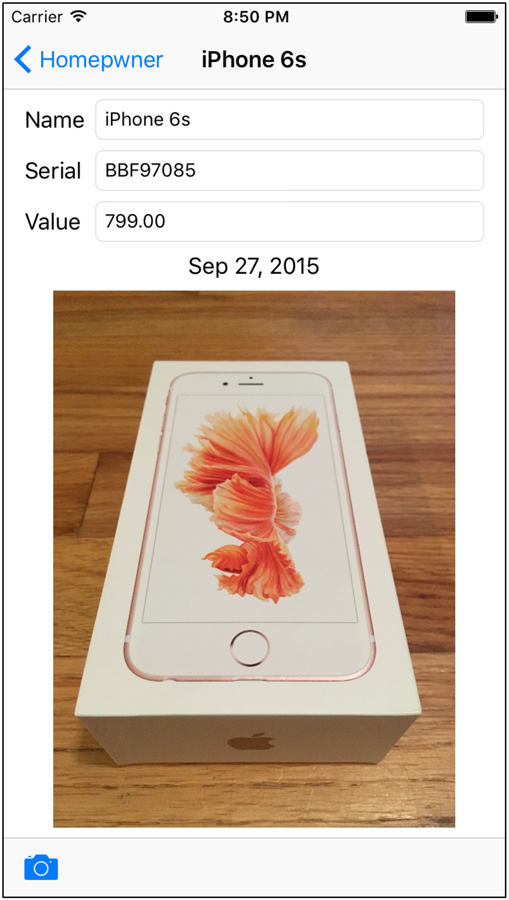
Вид может участвовать в двух отдельных цепей, одна в горизонтальном и в вертикальном направлении. К примеру, пользовательского интерфейса GeoQuiz из нашей книги Android можно создать с помощью ConstraintLayout двумя цепями, а не с помощью вложенных LinearLayouts:



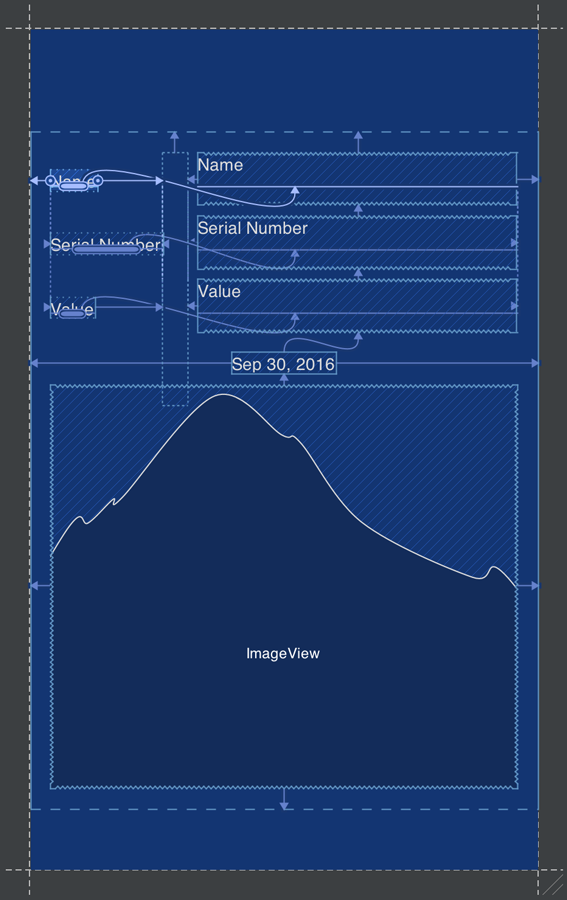
Пример: HomePwner

Используя StackViews является распространенным способом для упрощения ограничений на iOS. Например, чтобы создать форму, содержащую текстовые метки и текстовые поля, можно использовать вертикальные StackView , что содержит горизонтальный StackViewС UILabel и UITextField. Чтобы выровнять текстовые поля, мы бы добавили ведущим ограничением для всех из них.

В нашей [базе iOS Руководство по программированию](https://www.bignerdranch.com/we-write/ios-programming/), мы строим HomePwner приложения с деталью экран, который выглядит следующим образом:



Есть несколько вещей, чтобы иметь в виду при попытке повторить этот пользовательский интерфейс, используя ConstraintLayout. Поскольку параметры макета должны быть уникальными для каждого вида, мы не можем выровнять EditTexts друг с другом и установить их на минимальном расстоянии от соответствующего элемента. Одно из решений-ввести вспомогательную распорку вид и определение левой/правой ограничения ([макет XML-файл](https://www.bignerdranch.com/assets/img/blog/2016/12/homepwner_android.xml)):



Заключение

ConstraintLayout прошла долгий путь с момента ее первой Альфа-версии. Этот пост рассматривается несколько относительно простых пользовательских интерфейсов, которые могут быть построены с его помощью. Однако реальная обещание ConstraintLayout будут реализованы в более сложных пользовательских интерфейсов, которые обычно требуют нескольких слоев вложенных макетов, уменьшая глубину просмотра иерархии и, следовательно, повышение производительности разметки.

Вы бы хотели видеть больше примеров ConstraintLayout в действии? Дайте нам знать в комментариях!