Общие атрибуты View-элементов

Давайте рассмотрим фундаментальные характеристики любой View, вне зависимости от типа.

Итак, для того чтобы система могла отрисовать View, ей необходимо знать 2 составляющие – это размер и расположение.

Размер

Начнем с задания размеров. Для этого в .xml файле View задаются атрибуты **layout_height** и **layout_width**, и надеюсь понятно, что они являются обязательными для всех View.

Итак, высота и ширина могут быть выражены

- в конкретных значениях, например, 40dp
- **WRAP_CONTENT** – View займет столько места, сколько требуется для отрисовки его содержимого
- **MATCH_PARENT** – View займет все расстояние до краев его родителя.

Допустим, у нас есть 2 кнопки, стоящие друг рядом с другом. Одна кнопка занимает одну треть экрана, вторая – две трети. Как это сверстать? В этом случае нам нужно воспользоваться атрибутом **layout_weight** – так называемым весом, который специально создан для таких целей. Однако он работает только в **LinearLayout**. Вес определяет, сколько свободного места отойдет под ту или иную View. Атрибут направления, которое должно быть определено через вес, не игнорируется, а зануляется – 0dp.

В выше описанном дизайне, **layout_width** обоих кнопок будет **0dp**, а **layout_weight 1** и **2** соответственно.

Допустим, теперь нам нужно убрать первую кнопку, но оставить нетронутой размеры второй кнопки. Если мы просто удалим кнопку с разметки, то вторая кнопка займет всю ширину экрана, так как ей не с чем сравнивать свой атрибут веса. Чтобы избежать такой ситуации, в контейнер кнопки, то есть в LinearLayout, добавляется атрибут **weightSum** со значением **3**. Теперь, кнопка имеет правильный размер, но находится у левого края контейнера, тогда как раньше находилась у правого.

Расположение

Мы плавно подошли ко второй фундаментальной части любого элемента интерфейса, а именно – расположение. Мы использовали в качестве контейнера LinearLayout, который как нам уже известно, располагает элементы друг за другом. Можно воспользоваться атрибутом **layout_margin** и задать в численном виде отступ. Но этот атрибут скорее подходит для того, чтобы задавать отступы между View и от ближайшего края экрана. В нашем случае, он не подходит, так как мы не можем быть

уверены, что этот отступ подойдет для всех экранов.

Кстати, для указания отступов также можно воспользоваться атрибутом **padding**. Его отличие от марджина в том, что паддинг указывает отступ между границей View и ее содержимым.

Вернемся к проблеме. Попробуем воспользоваться гравитацией или атрибутами **gravity** и **layout_gravity**. Значения атрибутов означают направление, к которому View и будет притянута, они в принципе говорят сами за себя. В чем же разница между gravity и layout_gravity?

- **gravity** – это притяжение элементов **внутри текущего элемента**.
- **layout_gravity** – это притяжение текущего элемента **относительно родительского**.

То есть в нашем случае достаточно прописать **gravity = end** в нашем LinearLayout и кнопка выровняется относительно правого края. Конечно же, из-за своей специфики, сложно управлять притяжением в LinearLayout'е. Но чаще всего это и не требуется.

Также следует отметить атрибут **visibility**, который указывает как будет отрисовываться View. Варианты значений:

- **visible**, то есть View видимо на экране (по умолчанию)
- **invisible** – View невидимо, но занимает место на верстке
- **gone** – View невидимо и не занимает места

Этот атрибут позволяет добавлять гибкость интерфейсу, например, можно скрывать и показывать кнопку в зависимости от какого-либо условия. При переключении на **gone** и обратно, экран будет пересчитан и перерисован.

Стандартное отображение View можно попытаться изменить с помощью атрибута **background**. Как понятно из названия, этот атрибут задает фон View. В качестве значения можно использовать цвет, картинку или .xml файл с каким-нибудь векторным изображением, например, градиентом.