



План занятия


1. [Задача](#)
2. [UI](#)
3. [ConstraintLayout](#)
4. [Итоги](#)



ЗАДАЧА

ЗАДАЧА

Наша задача на сегодня —
подготовить layout (иногда говорят
макет или разметку) для поста нашей
социальной сети: →



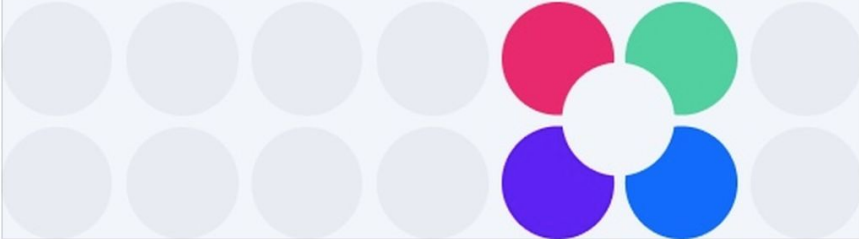
Нетология. Университет интернет-профессий
21 мая в 18:36

Привет, это новая Нетология!


Когда-то Нетология начиналась с интенсивов по онлайн-маркетингу. Затем появились курсы по дизайну, разработке, аналитике и управлению. Мы растём сами и помогаем расти студентам: от новичков до уверенных профессионалов.


Но самое важное остаётся с нами: мы верим, что в каждом уже есть сила, которая заставляет хотеть больше, целиться выше, бежать быстрее. Наша миссия — помочь встать на путь роста и начать цепочку перемен → <http://netolo.gy/fyb>


Привет!
Это новая :: нетология



Новая Нетология: 4 уровня карьеры
netology.ru

 22

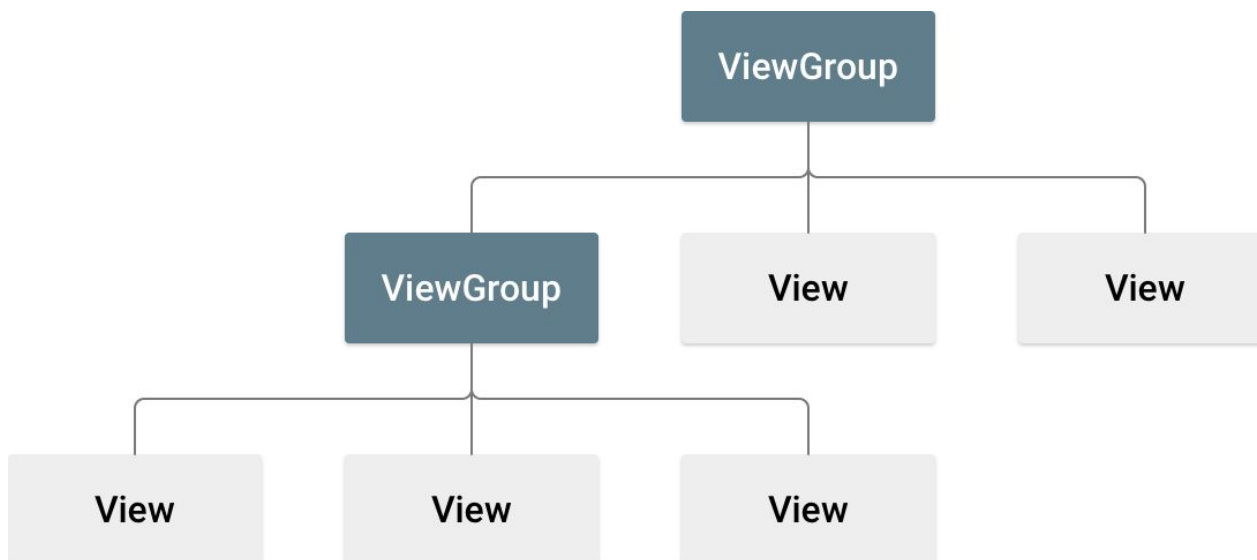
 1

 4.1K



UI

На прошлой лекции мы остановились на том, что есть View и ViewGroup и вместе они образуют иерархию вложенных друг в друга компонентов вида:



Изображение с [android.developer.com](https://developer.android.com)

Где под View понимаются компоненты, не наследуемые от ViewGroup.

LAYOUT

Наследники ViewGroup (будем называть их layout), позволяют размещать внутри себя другие View, являясь своеобразными контейнерами.

Layout достаточно много:

- FrameLayout,
- RelativeLayout,
- LinearLayout.

Вы можете найти целые руководства по их использованию.



LAYOUT

Указанные на предыдущем слайде layout подходят либо для простых макетов, либо требуют вкладывания одного layout в другой, усложняя иерархию интерфейса, что уменьшает производительность.

В современной разработке в большинстве случаев используют ConstraintLayout (и MotionLayout для анимаций), который позволяет не увеличивать иерархию, при этом гибко настраивать сам макет.

Именно его (а позже и его наследника — MotionLayout) мы и будем рассматривать.



КЛЮЧЕВОЕ

Стоит отметить, что в разработке макета (или компоновке) экранов нет единственно правильного или лучшего решения.

Поэтому стоит стремиться к:

- решению поставленной задачи,
- простоте,
- меньшему количеству кода.



CONSTRAINTLAYOUT



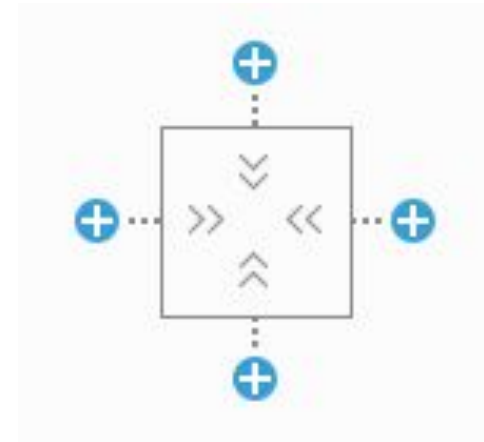
CONSTRAINTLAYOUT

[ConstraintLayout](#) поставляется в рамках Android Support Library (с которой вы должны были познакомиться в рамках ДЗ) и доступен, начиная с API v9.

BASIC CONSTRAINTS

Базовая идея: у каждого View есть 4 стороны (left, right, top и bottom), для которых мы можем установить ограничения (constraint):

- `layout_constraintTop_toTopOf`
- `layout_constraintTop_toBottomOf`
- `layout_constraintBottom_toTopOf`
- `layout_constraintBottom_toBottomOf`
- `layout_constraintBaseline_toBaselineOf`
- `layout_constraintStart_toEndOf`
- `layout_constraintStart_toStartOf`
- `layout_constraintEnd_toStartOf`
- `layout_constraintEnd_toEndOf`



START vs LEFT, END vs RIGHT

Q: если с Top всё более-менее понятно, то что такое Start и End?

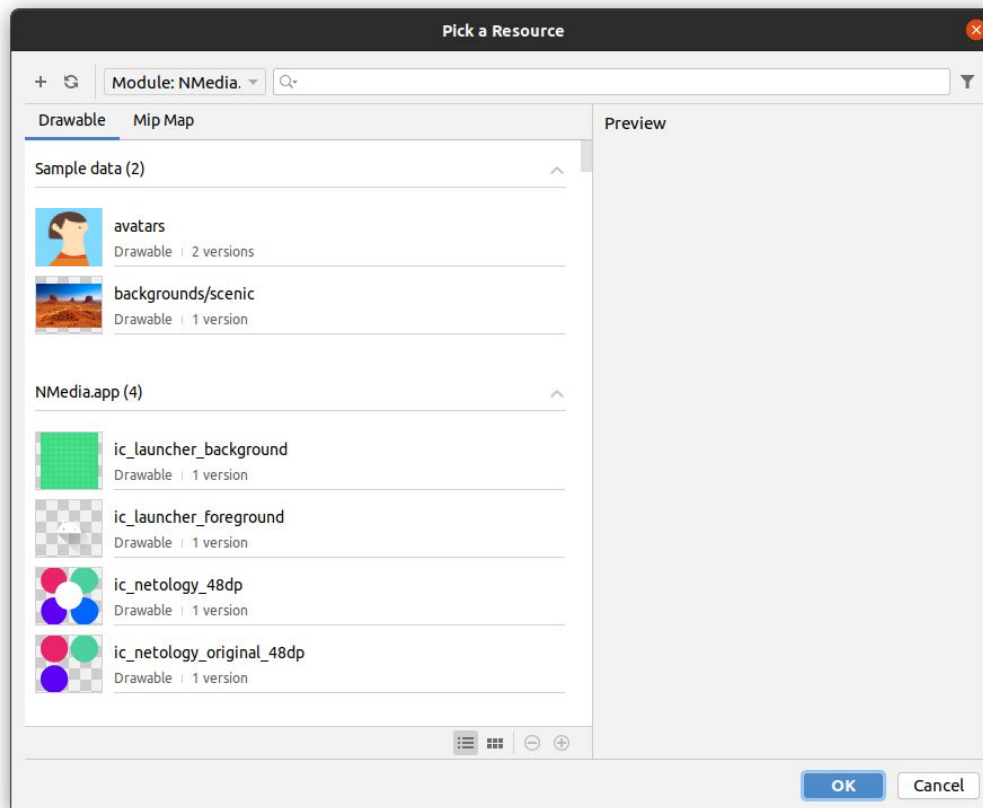
A: поскольку есть интерфейсы LTR (left to right) и RTL (right to left) в Android не рекомендуют использовать left и right, чтобы легко можно было затем обеспечить поддержку RTL-интерфейсов.

IMAGEVIEW



Нетология. Университет интернет-профессий
21 мая в 18:36

Начнём с аватарки: добавим на экран ImageView. Studio сразу предложит выбрать самую картинку:



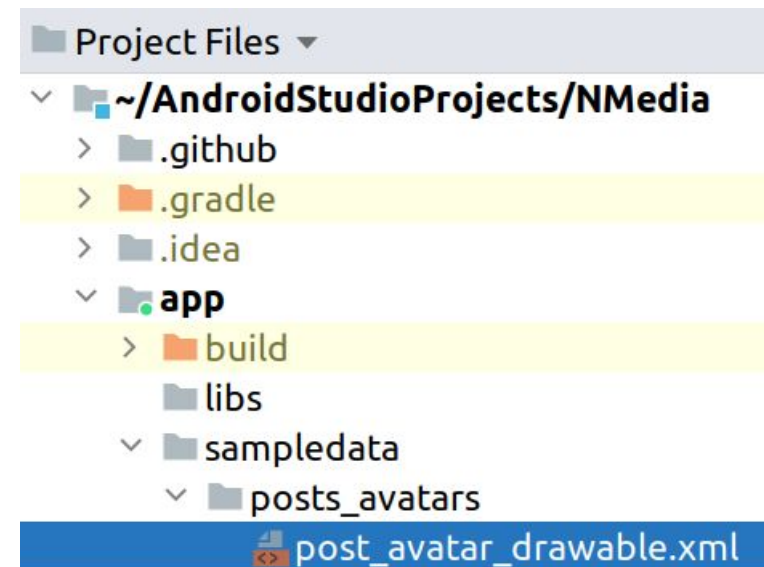
Варианта 3:

1. Sample Data
2. Ресурсы приложения
3. Ресурсы Android

РЕСУРСЫ

1. Sample Data — довольно смешные и далеки от реальной картинки.
2. Ресурсы приложения — не совсем подходят, т.к. аватарка будет грузиться с сервера, зачем нам «закидывать» лишние файлы?
3. Android — аналогично пункту 1.

Studio предлагает возможность создавать собственные демоданные. Эта возможность ещё развивается, но, если мы сделаем как на скриншоте, то сможем использовать собственные изображения (мы туда положили логотип Нетологии).



ГРУППЫ РЕСУРСОВ

В коде всё выглядит вот так:

```
<ImageView
    android:id="@+id/imageView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    tools:layout_editor_absoluteX="20dp"
    tools:layout_editor_absoluteY="17dp"
    tools:srcCompat="@sample/posts_avatars" />
```

зависит от того, куда поместили при перетаскивании

Давайте разбираться с префиксами:

- **android** — установка свойств компонента;
- **tools** — свойства для предпросмотра (не влияют на вид в приложении);
- **app** — свойства для Support Library (на скриншоте не представлены).

ID

id — идентификатор ресурса для компонента, который позволяет:

1. Ссылаться на него в layout.
2. Получать доступ к нему из кода.

При назначении (в форме `android:id`) всегда должен писаться с плюсом `@+id/name`. Символ «плюс» означает, что это новый ресурс и аарт сгенерирует для него новое поле в `R.java`.

CONSTRAINTS

Для того, чтобы «привязать» компонент, достаточно установить хотя бы два constraint: по вертикали и по горизонтали:

```
<ImageView
    android:id="@+id/imageView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginTop="16dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:srcCompat="@sample/posts_avatars" />
```



CONSTRAINTS

Чтобы понять основные возможности, достаточно пролистать [страницу документации, посвящённую ConstraintLayout](#).

MARGIN & PADDING

- `margin` — это отступ компонента снаружи (т.е. не входит в размеры компонента);
- `padding` — это отступ внутри компонента (т.е. входит в размер компонента).

Мы хотим, чтобы все компоненты внутри карточки поста имели определённый отступ от границы карточки. Мы, конечно, можем скрупулёзно выставлять `margin` у каждого дочернего компонента, а можем единоразово выставить `padding` у родителя. Можно использовать и другие варианты — какие, узнаем на следующих лекциях.

РАЗМЕРЫ

Напоминаем, что размеры могут быть:

- фиксированные в `dp`;
- относительно содержимого `wrap_content` (размер компонента автоматически выбирается так, чтобы всё его содержимое умещалось);
- относительно родителя `match_parent` (размер соответствует размеру родителя);
- специфический, определяемый функциональностью родителя (`match_constraint` для `ConstraintLayout`, записывается как `0dp`)

Для аватарки сейчас размер `48dp`, поскольку именно такая картинка у нас в `sample data`. Но лучше этот размер жёстко задать, чтобы какая бы картинка не пришла с сервера, её размер «не ломал» разметку.

РАЗМЕРЫ

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:padding="16dp"
    >
    <ImageView
        android:id="@+id/imageView"
        android:layout_width="16dp"
        android:layout_height="16dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:srcCompat="@sample/posts_avatars" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Studio вам всегда будет жёлтым выделять замечания. В данном случае рекомендуется задать описание для accessibility. Рекомендуем вам всегда устранять подобные замечания, чтобы код и ресурсы были чистыми.

ТЕКСТ

Далее — позиционирование текста:



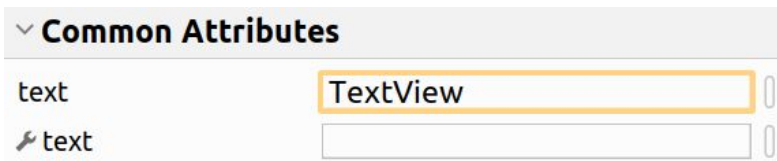
Нетология. Университет интернет-профессий
21 мая в 18:36

И здесь всё достаточно интересно: автор + дата совместно отцентрированы относительно аватарки.

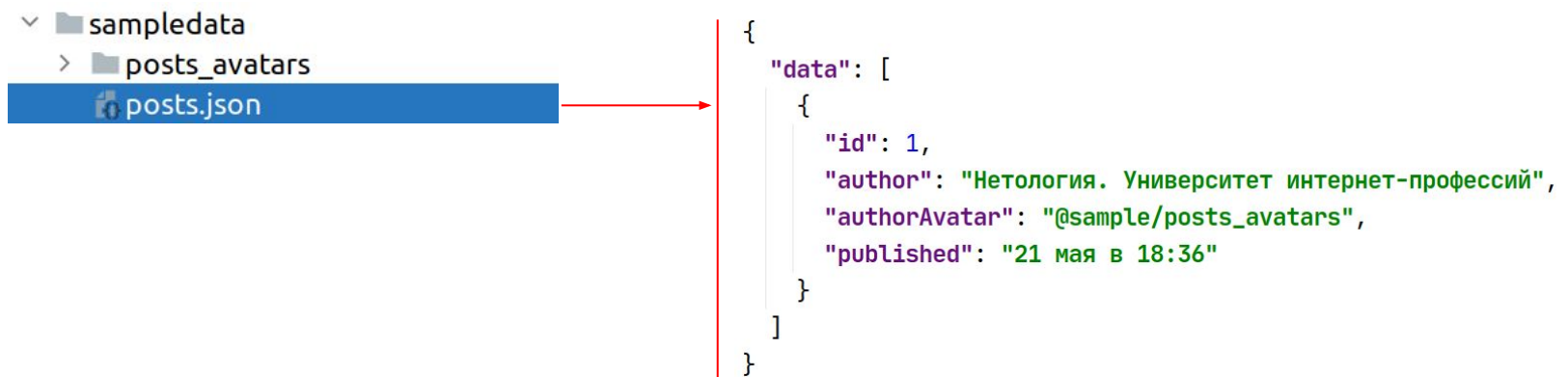
Создадим два `TextView` и попробуем их настроить.

TEXTVIEW

Обратите внимание, когда вы переносите из панели компонентов TextView, Studio автоматически выставляет text:



Его надо сразу убирать (только если это не фиксированный текст для вашего приложения) и заполнять tools:text (который с гаечным ключом):



TEXTVIEW

```
<ImageView
    android:id="@+id/imageView"
    android:layout_width="48dp"
    android:layout_height="48dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:srcCompat="@sample/posts.json/data/authorAvatar"
    android:contentDescription="@string/description_post_author_avatar" />
```

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    tools:text="@sample/posts.json/data/author" />
```

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    tools:text="@sample/posts.json/data/published" />
```

Чтобы продвинуться дальше, нам необходимо обсудить пару теоретических МОМЕНТОВ.

NAMING CONVENTION

Как именовать id? Например, аватар:

- `avatar_image_view`
- `avatarImageView`
- `avatarlv` (lv — сокращение от `ImageView`)
- `ivAvatar` (iv — сокращение от `ImageView`)
- `avatar`

NAMING CONVENTION

Правильного ответа здесь нет. Ключевое — консистентность и следование соглашениям, принятым в конкретном проекте и конкретной команде. Т.е. если вы выбрали один из вариантов — придерживайтесь его везде.

Мы выберем достаточно простой — avatar.

Q: почему?

A: просто потому, что если там сегодня `ImageView`, завтра может быть другой компонент, но это всё равно будет аватаром (хотя средства Studio позволяют легко его переименовывать).

ЦЕНТРИРОВАНИЕ

Начнём с центрирования:

1. Если хотим центрировать относительно экрана (вертикально), то верхний constraint привязываем к верхней границе родителя, нижний — к нижней границе родителя:

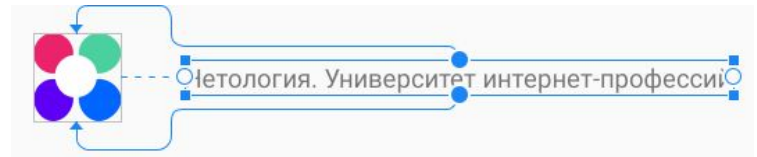
```
<TextView
    android:id="@+id/author"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:layout_editor_absoluteX="16dp"
    tools:text="@sample/posts.json/data/author" />
```

В Design Mode есть Helper (при клике на компоненте правой кнопкой мыши) — можно сразу выбрать центрирование.

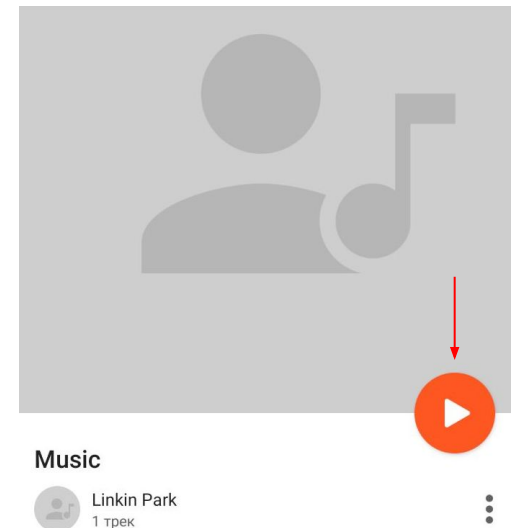
ЦЕНТРИРОВАНИЕ

2. Точно так же можно выровнять один компонент по центру другого:

```
<TextView  
    android:id="@+id/author"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    app:layout_constraintTop_toTopOf="@id/avatar"  
    app:layout_constraintBottom_toBottomOf="@id/avatar"  
    tools:text="@sample/posts.json/data/author" />
```



Точно так же можно центрировать по одной из границ (например, по нижней), чтобы получить эффект вида:

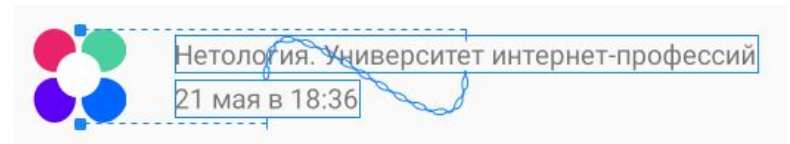


CHAINS

3. Chains позволяет «сцепить» два компонента по одной из осей, чтобы организовать из них некое подобие группы (при этом остальные constraint можно настраивать отдельно). В нашем случае, мы можем «сцепить» имя автора и время, и уже в таком виде отцентрировать относительно аватарки:

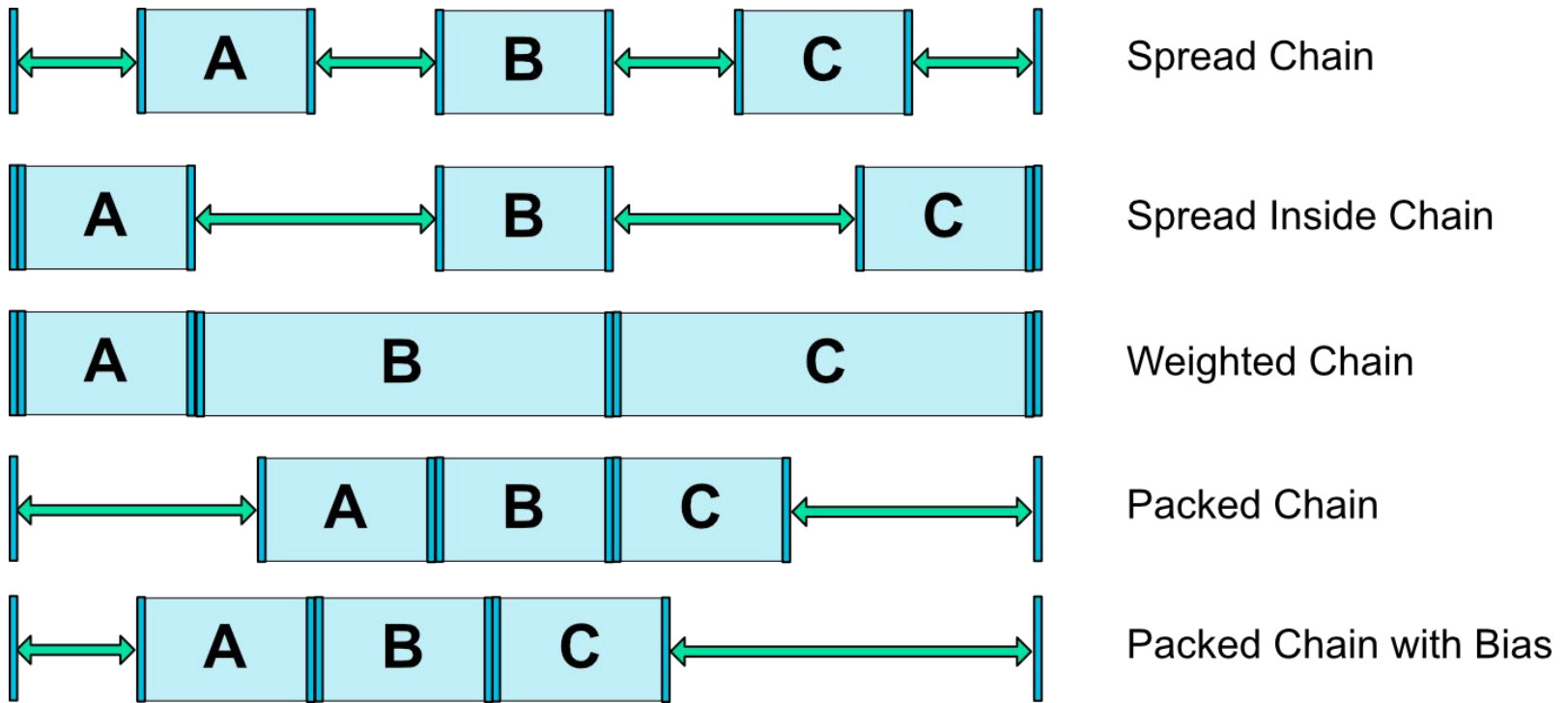
```
<TextView
    android:id="@+id/author"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toTopOf="@id/published"
    app:layout_constraintTop_toTopOf="@id/avatar"
    tools:text="@sample/posts.json/data/author" />
```

```
<TextView
    android:id="@+id/published"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="@id/avatar"
    app:layout_constraintTop_toBottomOf="@id/author"
    tools:text="@sample/posts.json/data/published" />
```



CHAINS

Chains — очень мощный инструмент, который также позволяет управлять «распределением» (chainStyle) сцепленных компонентов:



ВЫРАВНИВАНИЕ

А дальше достаточно выровнять левый край текстов с правым краем изображения и поставить необходимый отступ:

```
<TextView
    android:id="@+id/author"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toTopOf="@id/published"
    app:layout_constraintTop_toTopOf="@id/avatar"
    app:layout_constraintStart_toEndOf="@id/avatar"
    android:layout_marginStart="16dp"
    tools:text="@sample/posts.json/data/author" />
```

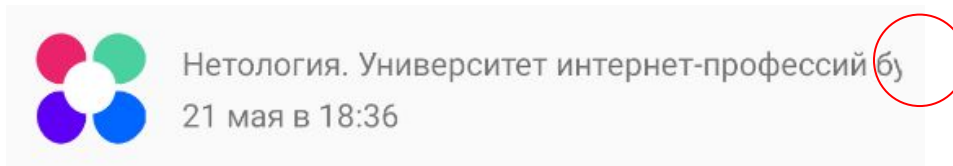
```
<TextView
    android:id="@+id/published"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="@id/avatar"
    app:layout_constraintTop_toBottomOf="@id/author"
    app:layout_constraintStart_toEndOf="@id/avatar"
    android:layout_marginStart="16dp"
    tools:text="@sample/posts.json/data/published" />
```



Нетология. Университет интернет-профессий
21 мая в 18:36

ТЕКСТ

Теперь ключевой момент, о который спотыкаются все начинающие разработчики и дизайнеры — объём текста. Сейчас название автора уместается на экран. Но что будет, если оно станет немного больше?



Обычно вам показывают дизайн до того, как вы возьмётесь за работу, поэтому обязательно спрашивайте дизайнера, как должно выглядеть приложение.

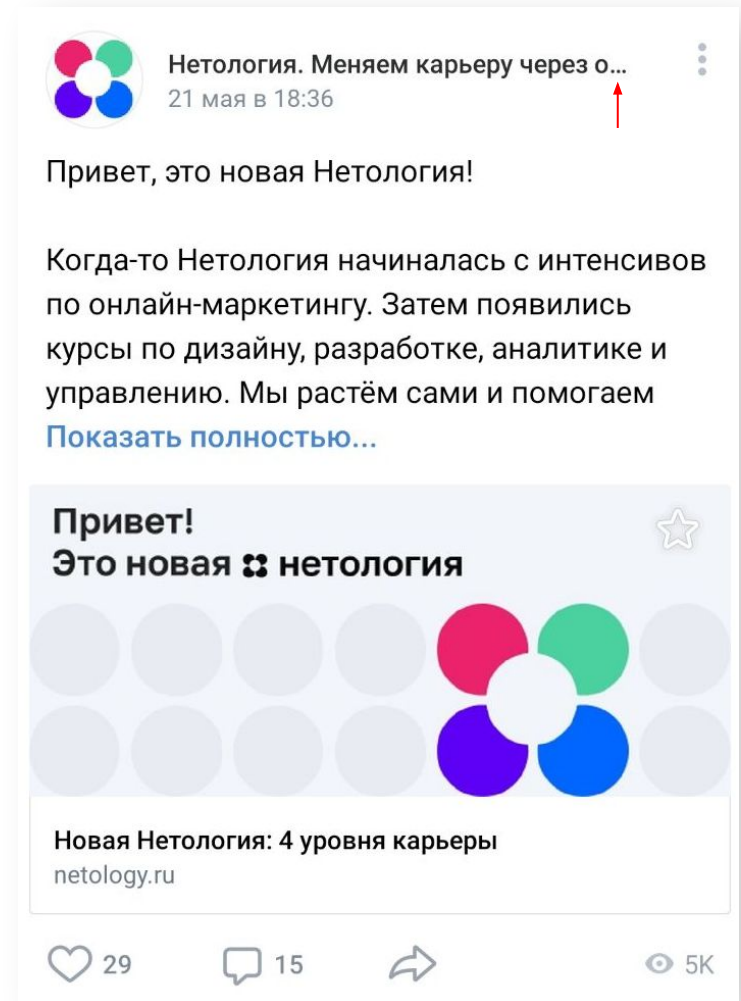
?

Как вы думаете, как обычно поступают в случаях, если текст не вмещается на экран?

ТЕКСТ

Мы специально показали вам веб-версию. Мобильная версия выглядит следующим образом:

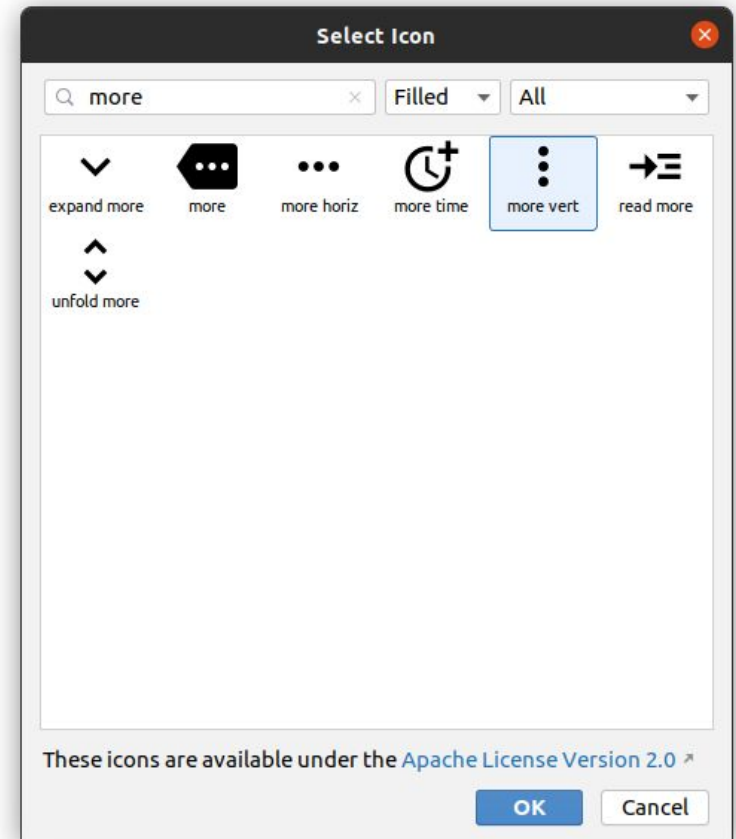
Т.е. текст идёт не впритык, а остаётся место. Текст, который не поместился заменяется на «...».



ТЕКСТ

Иконку с тремя точками мы можем добавить через Vector Asset Studio (clip art называется more vert):

Ключевой вопрос: какого размера должна быть иконка?

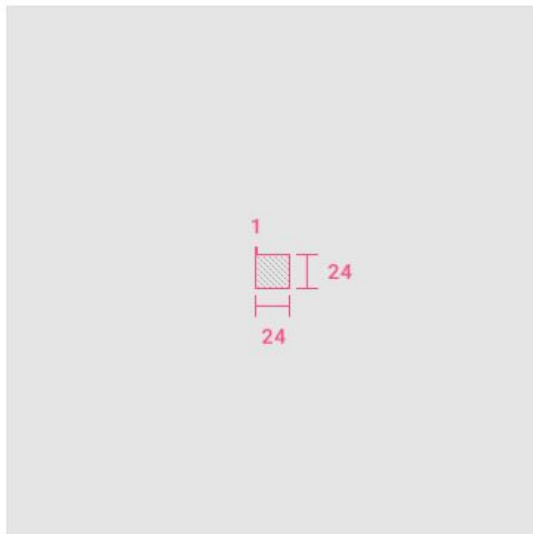


MATERIAL DESIGN

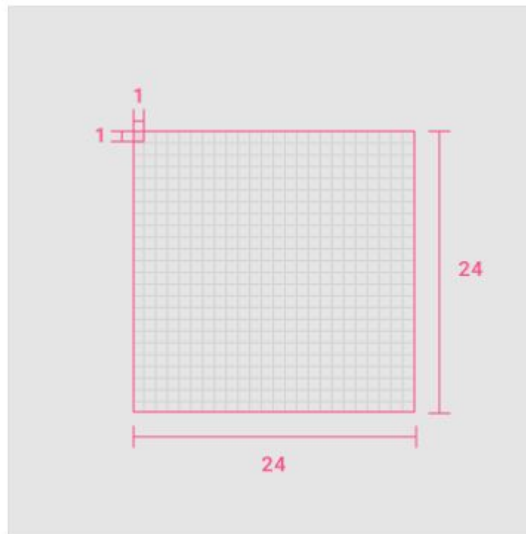
Android использует Material Design и именно туда стоит «подглядывать», когда речь идёт о размерах, о том, как стоит и как не стоит делать (хотя у многих дизайнеров с этим большая проблема):

Icon sizes

System icons are displayed as 24 x 24 dp. Create icons for viewing at 100% scale for pixel-perfect accuracy.



100% scale

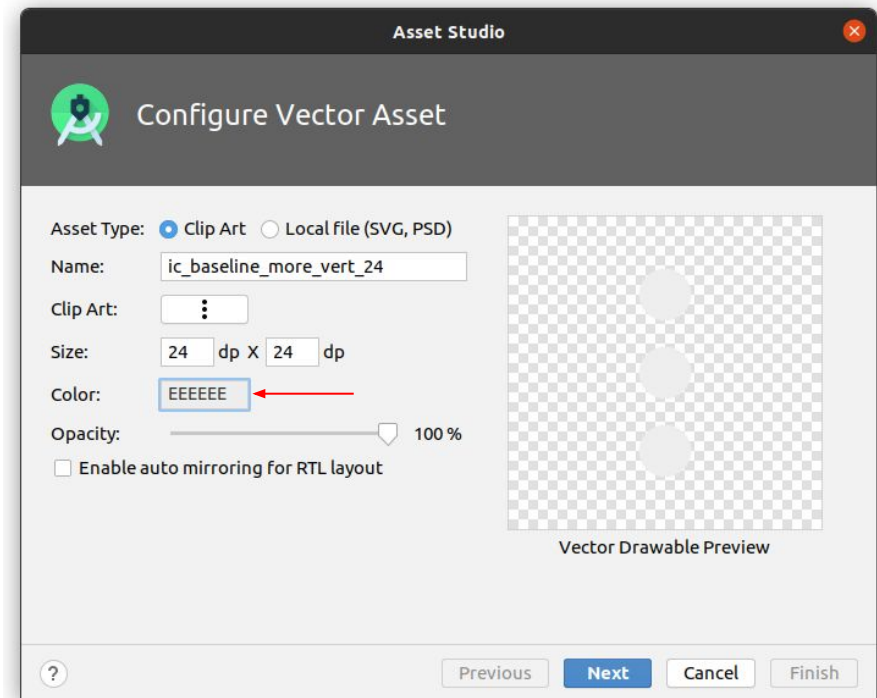


1000% scale

COLOR

Цвет иконки можно задать тут же с помощью цветовой модели RGB:

Кстати, его можно будет поменять потом прямо в XML.



RGB, RGBA

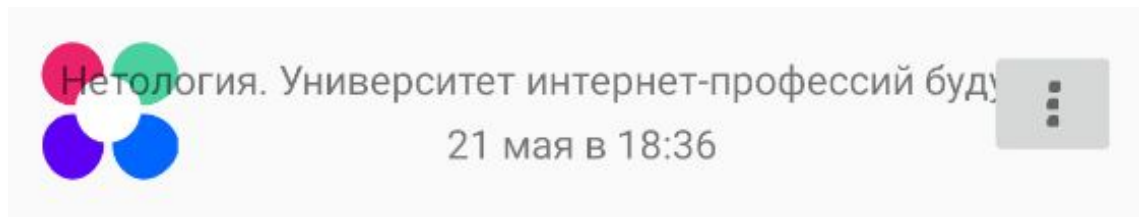
RGB — это цветовая модель, которая позволяет закодировать цвет с помощью трёх компонентов: Red, Green, Blue. Эти цвета можно представить в диапазоне от 0 до 255, что в шестнадцатеричном виде записывается как от 00 до FF.

Итого вся тройка может быть закодирована вот таким числом: #FFFFFF (белый) или #000000 (чёрный).

RGBA — дополнительно к RGB добавляет альфа-канал (прозрачность), которая тоже меняется от 0 до 255. #FF000000 — абсолютно непрозрачный чёрный, #80000000 — чёрный с 50% прозрачностью.

IMAGEBUTTON

Если мы поместим компонент ImageButton и выставим все Constraints так же, как делали для аватарки, то получим «плачевную» картину:

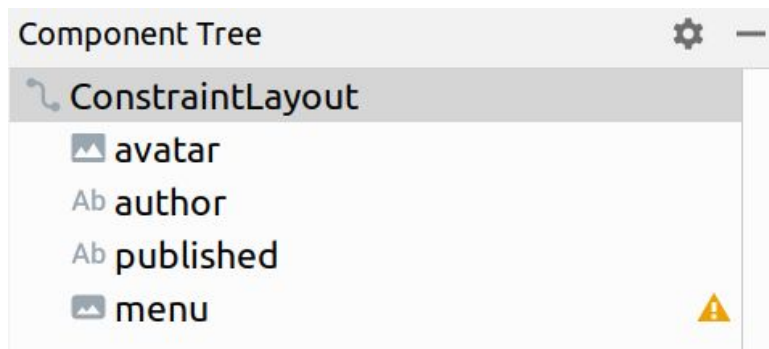


Во-первых, текст не обрезается, а заезжает на соседние компоненты (обратите внимание, только на аватар). *Как вы думаете — почему?*

Во-вторых, текст стал центрироваться. *Как вы думаете — почему?*

НАЛОЖЕНИЕ

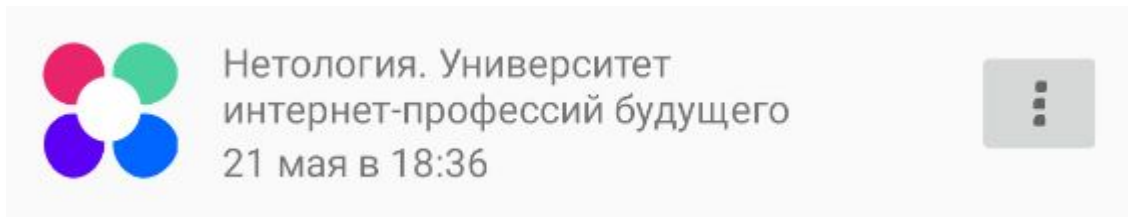
Текст наезжал на аватар потому, что в нашем layout аватар выше, чем текст. В графических интерфейсах нижестоящий как бы «накладывается» сверху на вышестоящий:



ось Z

ЦЕНТРИРОВАНИЕ ТЕКСТА

Проблема в том, что у текста выставлено `wrap_content` и два `constraint`, что и приводит к выравниванию. Вместо этого мы можем поставить `0dp (match_constraint)`:

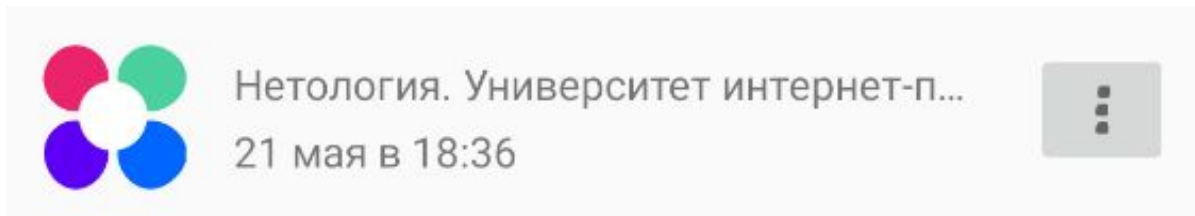


Стало лучше, но не намного, теперь текст пишется в две строки.

ОБРЕЗАНИЕ ТЕКСТА

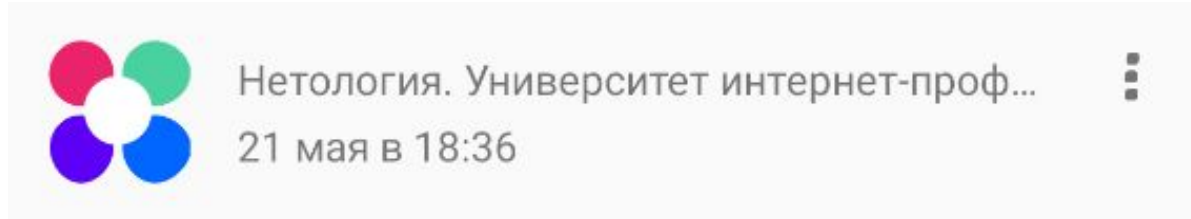
Чтобы достичь желаемого эффекта, нам нужно выставить два атрибута:

- `singleLine`
- `ellipsize`



ФОН КНОПКИ

Теперь разберёмся с фоном кнопки: установим фон в прозрачный
`background="@android:color/transparent"`:



ССЫЛКИ НА РЕСУРСЫ

Q: почему мы до этого использовали запись `@drawable/имя_ресурса`, а теперь `@android:color/transparent` (с `android:`)?


A: То, что лежит в каталоге `res` нашего приложения начинается сразу с типа ресурса `@тип_ресурса`, а то, что в «Android» — с префикса `@android:тип_ресурса`.

Из кода это будет `R.drawable` или `android.R.drawable`.

«ОСТАТКИ»

Давайте подумаем, что делать с оставшимися частями:

1. Текстом.
2. Блоком ссылки.
3. Нижним блоком с лайками, репостами и просмотрами.

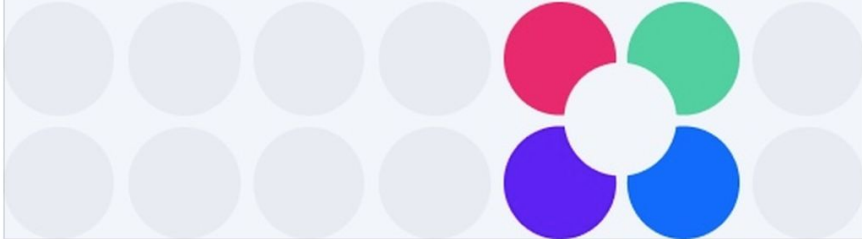
**Нетология. Университет интернет-профессий**
21 мая в 18:36

Привет, это новая Нетология!




Когда-то Нетология начиналась с интенсивов по онлайн-маркетингу. Затем появились курсы по дизайну, разработке, аналитике и управлению. Мы растём сами и помогаем расти студентам: от новичков до уверенных профессионалов.

Но самое важное остаётся с нами: мы верим, что в каждом уже есть сила, которая заставляет хотеть больше, целиться выше, бежать быстрее. Наша миссия — помочь встать на путь роста и начать цепочку перемен → <http://netolo.gy/fyb>

Привет!
Это новая :: нетология



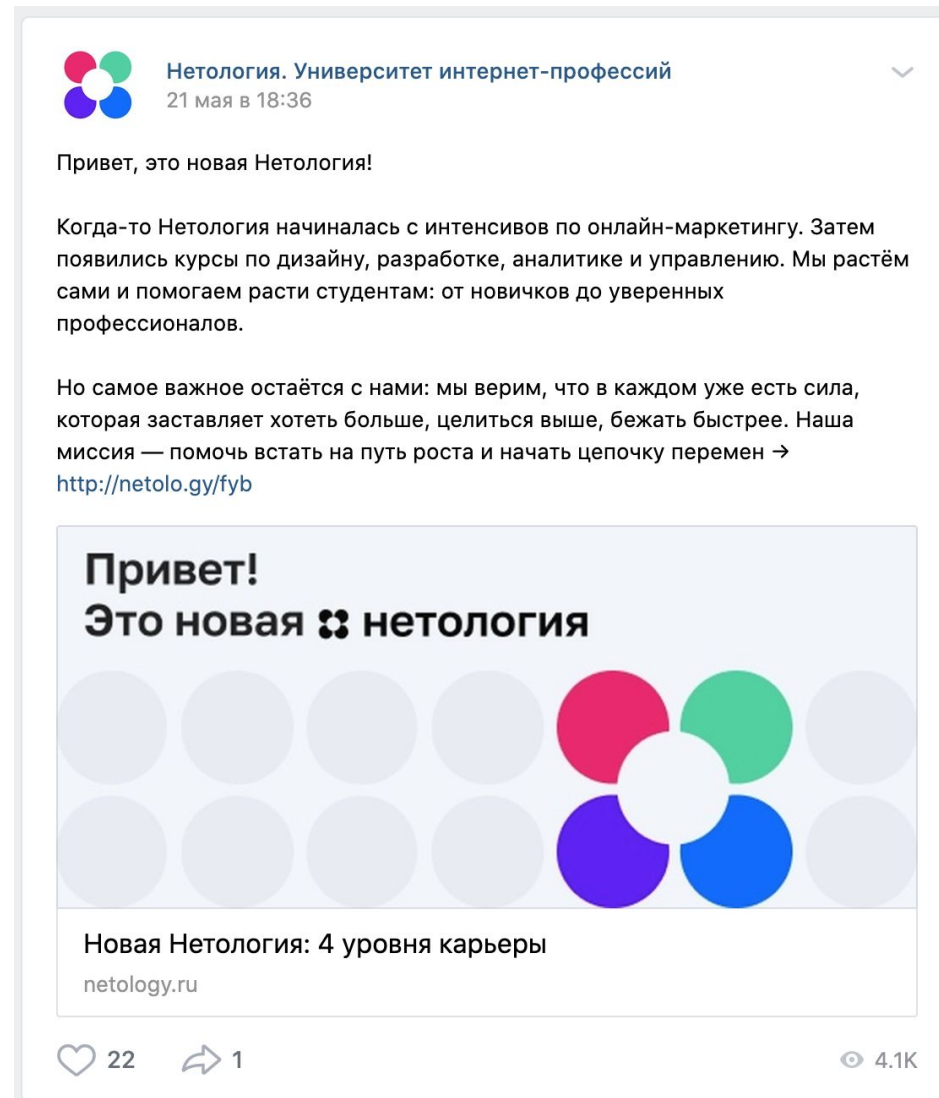
Новая Нетология: 4 уровня карьеры
netology.ru

 22  1  4.1K

«ОСТАТКИ»

По факту:


1. Текст — это просто TextView, в виде vertical chain с блоком ссылки и нижним блоком.
2. Блок ссылки — это vertical chain из изображения и двух TextView.
3. Нижний блок — просто привязанные к разным краям компоненты.



«ОСТАТКИ»

Мы специально выделили блок ссылки как отдельный компонент. Если мы возьмём несколько постов, то общая часть у них будет одинакова, а вот наличие или отсутствие ссылки будет определять наличие этого блока.

Поэтому будет логично вынести его в отдельный файл и просто подключать (как это сделать, мы разберём на следующих лекциях профессии).



Нетология. Университет интернет-профессий

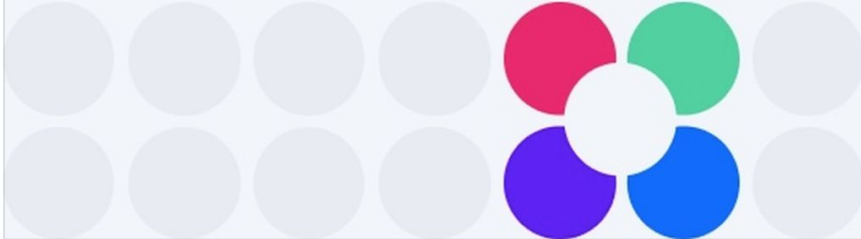
21 мая в 18:36

Привет, это новая Нетология!

Когда-то Нетология начиналась с интенсивов по онлайн-маркетингу. Затем появились курсы по дизайну, разработке, аналитике и управлению. Мы растём сами и помогаем расти студентам: от новичков до уверенных профессионалов.


Но самое важное остаётся с нами: мы верим, что в каждом уже есть сила, которая заставляет хотеть больше, целиться выше, бежать быстрее. Наша миссия — помочь встать на путь роста и начать цепочку перемен → <http://netology.ru>


Привет!
Это новая :: нетология




Новая Нетология: 4 уровня карьеры

netology.ru

 22

 1

 4.1K

РАЗМЕРЫ

Вы уже, наверное, заметили, что размеры вроде 16dp фигурируют достаточно часто и неплохо бы их вынести в отдельный файл и на них ссылаться (как мы делаем со всеми ресурсами):

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="common_spacing">16dp</dimen>
    <dimen name="posts_avatar_size">48dp</dimen>
</resources>
```

```
<ImageView
    android:id="@+id/avatar"
    android:layout_width="@dimen/posts_avatar_size"
    android:layout_height="@dimen/posts_avatar_size"
    android:contentDescription="@string/description_post_author_avatar"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:srcCompat="@sample/posts.json/data/authorAvatar" />
```

```
<TextView
    android:id="@+id/author"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="@dimen/common_spacing"
    android:layout_marginEnd="@dimen/common_spacing"
    android:ellipsize="end"
    android:singleLine="true"
    app:layout_constraintBottom_toTopOf="@id/published"
    app:layout_constraintEnd_toStartOf="@id/menu"
    app:layout_constraintStart_toEndOf="@id/avatar"
    app:layout_constraintTop_toTopOf="@id/avatar"
    tools:text="@sample/posts.json/data/author" />
```

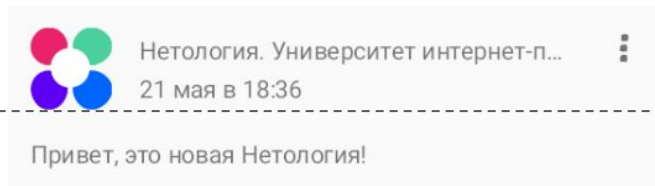


РАЗМЕРЫ

Стоит отметить, что в лекции по Material Design мы с вами детально разберём, какие размеры, у каких компонентов должны быть.

BARRIER

Теперь самая интересная часть — основной текст. Он должен быть ниже аватара, даты публикации и иконки опций :



Почему мы указали, что ниже именно всех трёх? Потому что если дизайн немного изменится, то один из компонентов может оказаться ниже (аватар или текст).

При этом `ConstraintLayout` позволяет указать ограничения по одной стороне только относительно одного другого компонента (нельзя написать

`layout_constraintTop_toBottomOf="@id/avatar,@id/published")`.

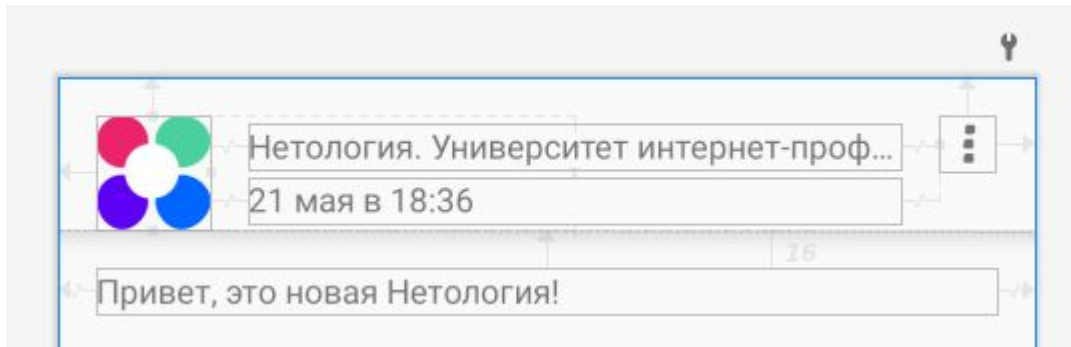
BARRIER

Именно для этих целей ConstraintLayout предлагает специальный компонент — [Barrier](#). Barrier позволяет выбрать несколько компонентов и создать виртуальную линию, которая будет двигаться в зависимости от размеров выбранных компонентов (а к этой линии уже можно привязывать другие компоненты):



BARRIER

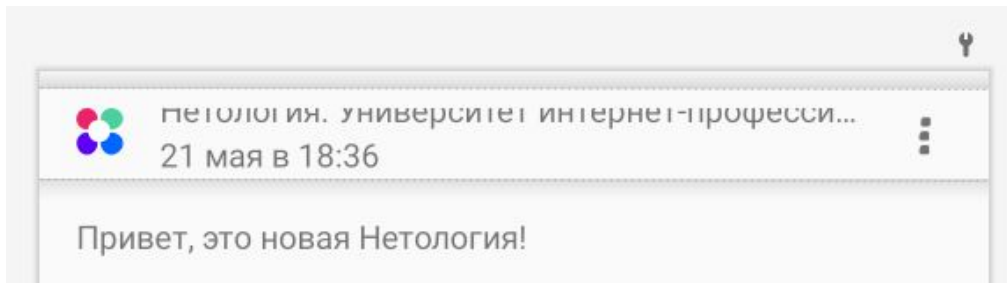
Мы можем установить горизонтальный barrier и установить свойство `barrierDirection="bottom"` — барьер будет двигать всё вниз:



Общее правило: если подряд идут несколько виджетов и вы хотите привязаться к общей границе (не важно, какой из них ниже/выше), то вы используете barrier.

BARRIER

Если barrier помог нам при увеличении размера аватара, то вот при уменьшении получается вот такая беда:

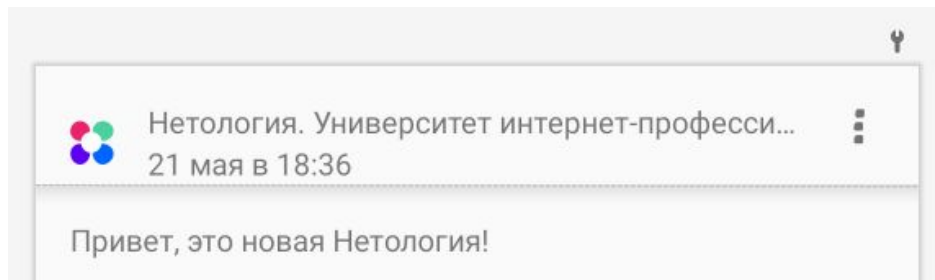


Q: но зачем мы смотрим на уменьшение аватара? Ведь размер аватара фиксирован?

A: обычно это бывает так: один из ваших коллег решает «чуть-чуть» поменять размер аватара (он же хранится в dimensions) и в итоге все экраны разваливаются.

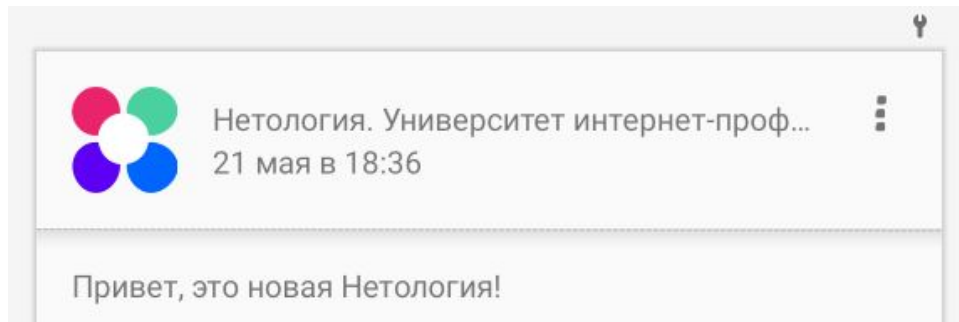
BARRIER

Мы, конечно, можем поставить Barrier и сверху, но смысла нет. Сверху у нас и так пролегает граница по родителю (parent), от которой мы и можем оттолкнуться, центрировав аватар и chain по вертикали:



BARRIER

Ну и, конечно, нужно не забыть добавить отступы по вертикали (как к аватару, так и к дате публикации):



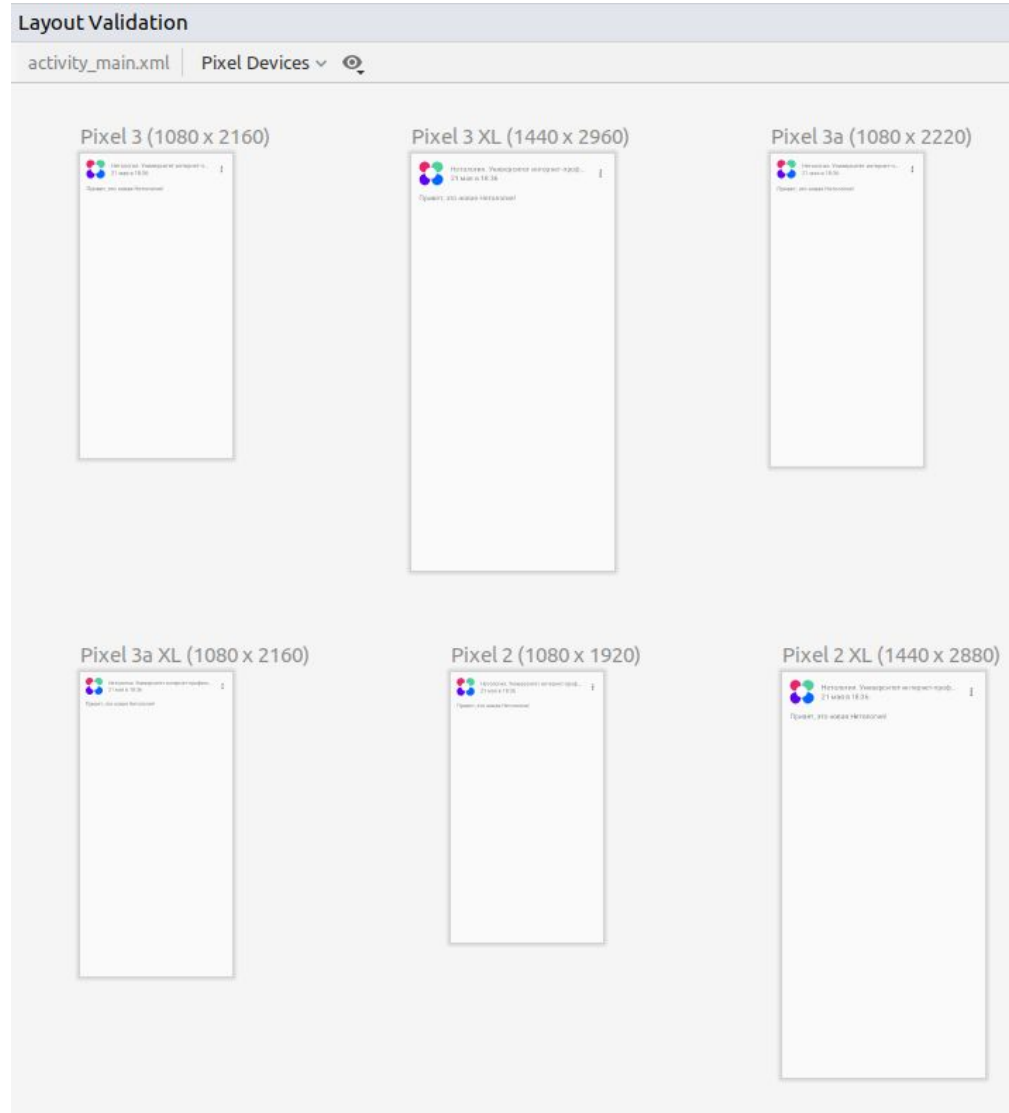


КЛЮЧЕВОЕ

Ключевое: не стремитесь сразу использовать все возможности ConstraintLayout и все вспомогательные компоненты, которые он предоставляет. Начните с простых constraint.

КЛЮЧЕВОЕ

Обязательно проверяйте
макет с помощью Layout
Validation:





ИТОГИ



ИТОГИ

Сегодня мы обсудили базовые вопросы работы с `ConstraintLayout` — одним из самых распространённых «контейнеров» для организации UI.

Конечно же, возможностей у него гораздо больше, чем мы успели рассмотреть на лекции (особенно после обновления до версии 2.0), и мы разберёмся с ними в следующих лекциях и ДЗ.