



План занятия

1. [О переводах](#)
2. [Ресурсы \(обзор\)](#)
3. [Локализация](#)
4. [Qualifiers](#)
5. [UI](#)
6. [Итоги](#)



О ПЕРЕВОДАХ



О ПЕРЕВОДАХ

Чем больше мы будем работать непосредственно с Android, тем больше мы будем сталкиваться с различными терминами.

Проблем с этим две:

1. Достаточно сложно их перевести на русский язык (либо при переводе искажается смысл).
2. Даже если их перевести, то русскоязычный эквивалент используется настолько редко, что вас попросту не поймут в разговоре.



О ПЕРЕВОДАХ

Поэтому в лекциях, мы будем использовать по большей части именно англоязычные термины: например, `activity`, `intent filter` вместо «активность» или «операция» и «фильтр намерений».

Но в то же время там, где это оправдано, будем употреблять совместно цвет и размер вместе с `color` и `dimension`.



РЕСУРСЫ. ОБЗОР

РЕСУРСЫ

Ресурсы — статичные файлы, которые располагаются в каталоге [res](#).

Ключевых целей выделения ресурсов в отдельные файлы две:

1. «Разделяй и властвуй»: код (логика) — отдельно, ресурсы — отдельно (можно отдать переводчику языковой файл, дизайнеру — изображения, при этом код не изменится).
2. Android подгружает нужные ресурсы в зависимости от конфигурации устройства (при этом снимая необходимость «программирования» этой логики).

РЕСУРСЫ

Ресурсы представлены в двух ключевых форматах:

- XML — общий формат описания, используемый Android;
- специфичный для типа: svg, png, jpg — изображения, ttf, otf — шрифты и т.д.

ГРУППЫ РЕСУРСОВ

Ресурсы объединены в группы:

- animator, anim — анимации;
- color — изменяемые в зависимости от состояния цвета;
- drawable — изображения и «фигуры»;
- mipmap — иконки запуска;
- layout — разметка UI;
- menu — пункты меню;
- raw — файлы, не интерпретируемые специальным образом;
- values — значения (цвета, размеры, строки, стили);
- XML — произвольные данные в формате XML;
- font — шрифты.

обрабатываются как поток байт



-
- наиболее используемые




VALUES

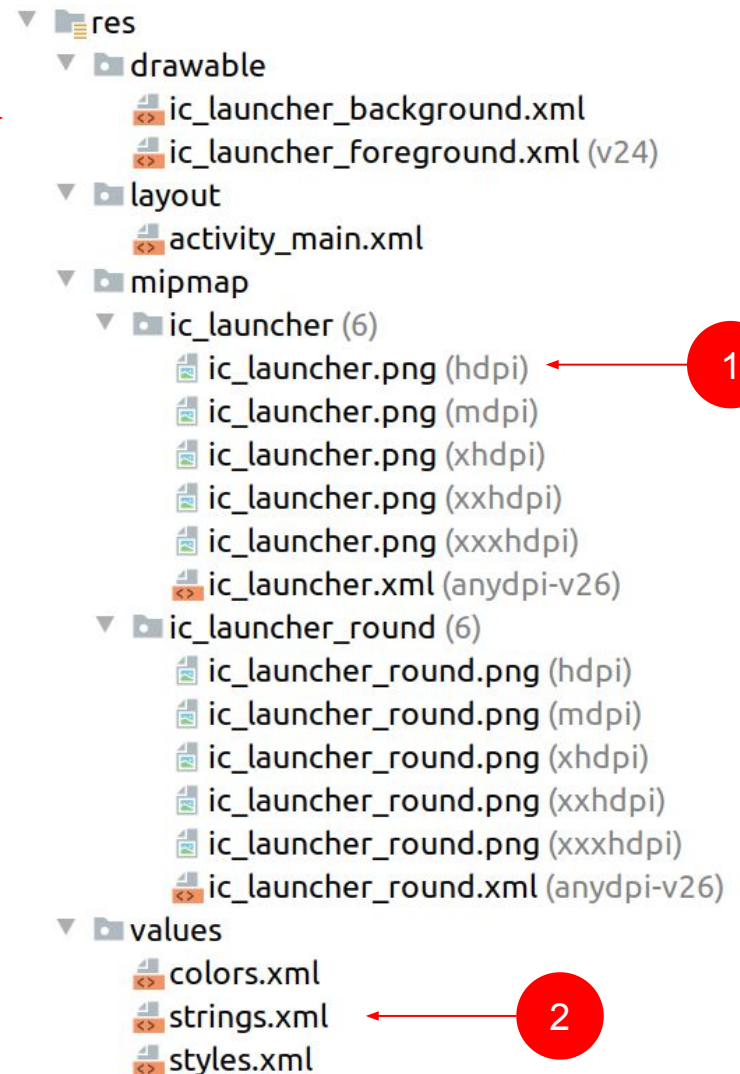
Values — это каталог, в котором размещены файлы, содержащие следующие типы:

- array — массивы значений;
- bool — boolean значения;
- color — цвета;
- dimension — размеры;
- id — идентификаторы;
- string — строки;
- style — стили.

РЕСУРСЫ

По умолчанию генерируются
конструктором : 

Обратите внимание: некоторые
ресурсы в скобках имеют
надпись (1), а другие — нет (2).



Вид Android

- ▼ res
 - ▼ drawable
 - ic_launcher_background.xml
 - ic_launcher_foreground.xml (v24)
 - ▼ layout
 - activity_main.xml
 - ▼ mipmap
 - ▼ ic_launcher (6)
 - ic_launcher.png (hdpi)
 - ic_launcher.png (mdpi)
 - ic_launcher.png (xhdpi)
 - ic_launcher.png (xxhdpi)
 - ic_launcher.png (xxxhdpi)
 - ic_launcher.xml (anydpi-v26)
 - ▼ ic_launcher_round (6)
 - ic_launcher_round.png (hdpi)
 - ic_launcher_round.png (mdpi)
 - ic_launcher_round.png (xhdpi)
 - ic_launcher_round.png (xxhdpi)
 - ic_launcher_round.png (xxxhdpi)
 - ic_launcher_round.xml (anydpi-v26)
 - ▼ values
 - colors.xml
 - strings.xml
 - styles.xml

Вид Project

- ▼ res
 - ▼ drawable
 - ic_launcher_background.xml
 - ▼ drawable-v24
 - ic_launcher_foreground.xml
 - ▼ layout
 - activity_main.xml
 - ▼ mipmap-anydpi-v26
 - ic_launcher.xml
 - ic_launcher_round.xml
 - ▼ mipmap-hdpi ← каталоги с суффиксом
 - ic_launcher.png
 - ic_launcher_round.png
 - ▼ mipmap-mdpi
 - ic_launcher.png
 - ic_launcher_round.png
 - ▼ mipmap-xhdpi
 - ic_launcher.png
 - ic_launcher_round.png
 - ▼ mipmap-xxhdpi
 - ic_launcher.png
 - ic_launcher_round.png
 - ▼ mipmap-xxxhdpi
 - ic_launcher.png
 - ic_launcher_round.png
 - ▼ values
 - colors.xml
 - strings.xml
 - styles.xml

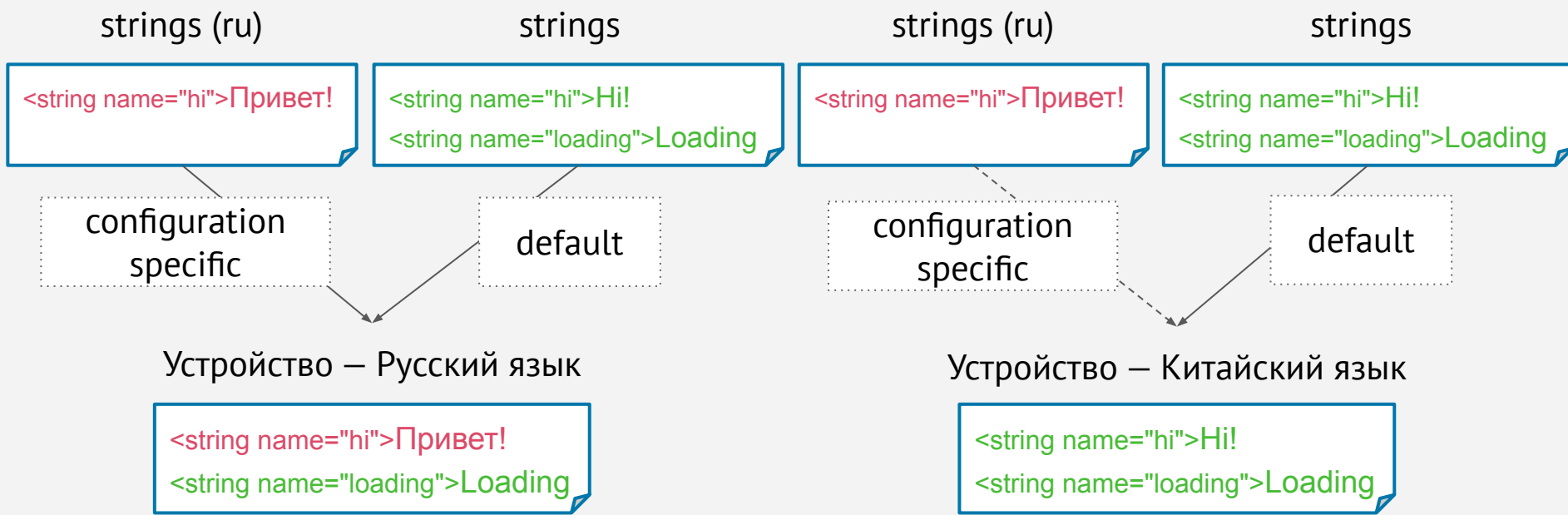
DEFAULT vs CONFIGURATION SPECIFIC

В зависимости от того, есть у каталога суффикс или нет:

- с суффиксом — configuration specific;
- без суффикса — default.

DEFAULT vs CONFIGURATION SPECIFIC

Общая логика следующая: когда текущая конфигурация устройства соответствует суффиксу, тогда загружается configuration specific, переопределяя значения, указанные в default.





ЛОКАЛИЗАЦИЯ

ЯЗЫК

Начнём с языка. Все фразы должны быть вынесены из кода и layout в файлы strings:

```
<TextView
```

```
    android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
```

```
    android:text="Hello World!"
```

Extract string resource

Override Resource in Other Configuration...

Rearrange tag attributes

Remove attribute

Inject language or reference

Extract Resource

Resource name: hello

Resource value: Hello World!

Source set: main

File name: strings.xml

Create the resource in directories:

☒ values

OK

Cancel

ЯЗЫК

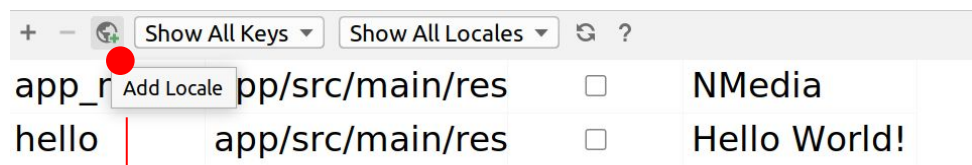
В layout вместо конкретной строки будет прописан id в формате `@string/hello`, где `@string` — тип ресурса (не название файла, а тип ресурса), а `hello` — имя:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

```
<resources>
    <string name="app_name">NMedia</string>
    <string name="hello">Hello World!</string>
</resources>
```


ЯЗЫК

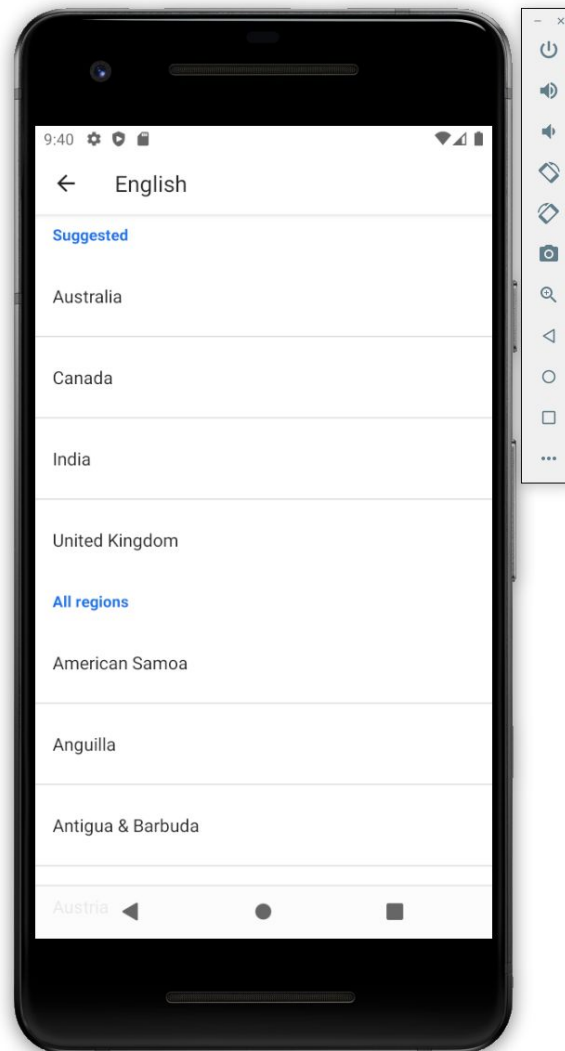
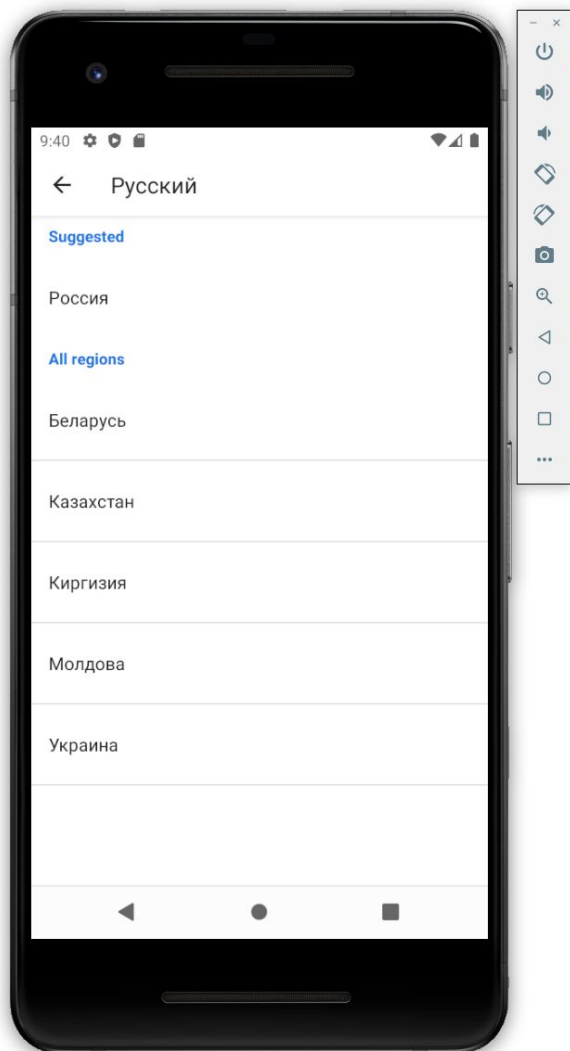
Добавим перевод. Сделать это можно через Translation Editor:



- Russian (ru)
- Russian (ru) in Belarus (BY)
- Russian (ru) in Kazakhstan (KZ)
- Russian (ru) in Kyrgyzstan (KG)
- Russian (ru) in Moldova (MD)
- Russian (ru) in Russia (RU)
- Russian (ru) in Ukraine (UA)

Key	Resource Folder	Untranslatable	Default Value	Russian (ru)
app_name	app/src/main/res	<input type="checkbox"/>	NMedia	NMedia
hello	app/src/main/res	<input type="checkbox"/>	Hello World!	

ЯЗЫК



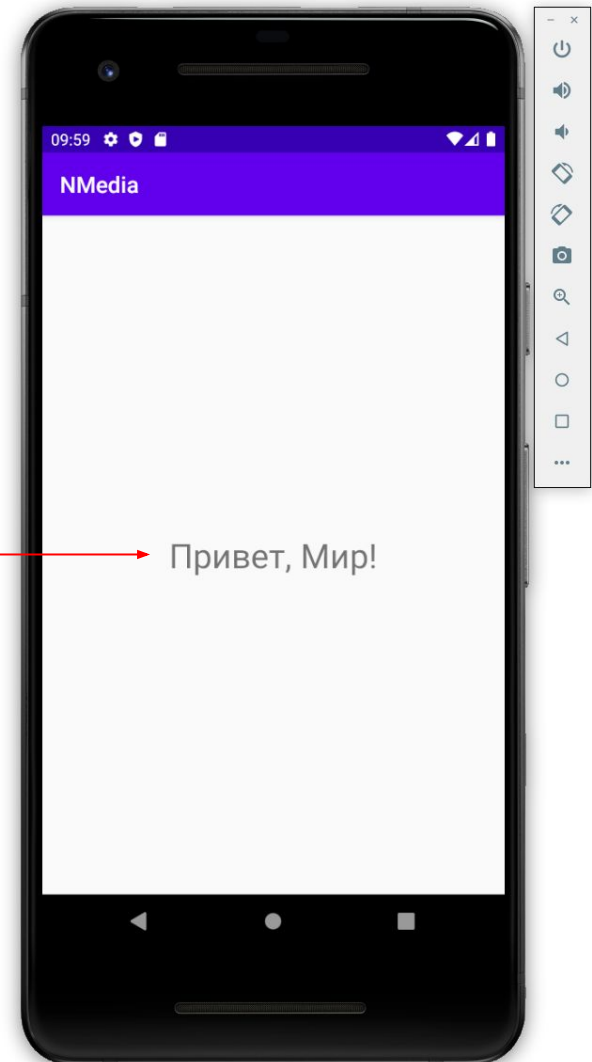
Вы будете
использовать
только язык
без региона,
но для
английского
часто
выделяют
en-US и en-GB.

ЯЗЫК

Если теперь добавить

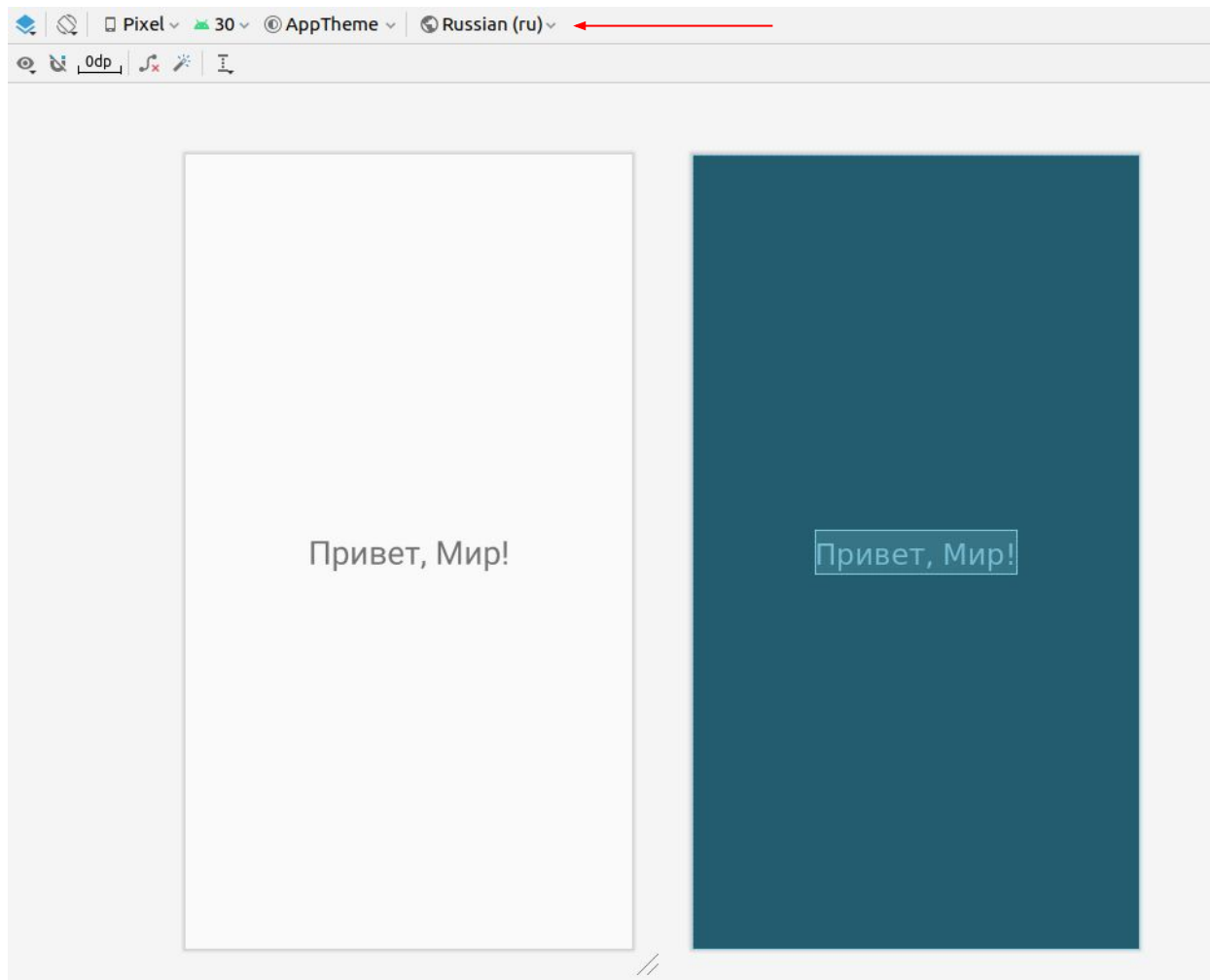
`<string name="hello">Привет, Мир!</string>`

в strings (ru), то увидим:





DESIGNER

Не обязательно каждый раз запускать приложение:





ДОСТУП ИЗ КОДА

Как вы уже знаете, для ресурсов формируется класс **R**, поэтому:

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
        println(R.string.hello) // число  
        println(getString(R.string.hello)) // "Привет, Мир!" или "Hello World!"  
    }  
}
```

Чтобы быстро применить изменения (для эмуляторов Android 8.0+):

-  применить изменения ресурсов и кода и перезапустить Activity
-  применить изменения кода (ресурсы перезагружаться не будут)

Logcat:

2020-08-20 10:09:00.774 ru.netology.nmedia I/System.out: 2131492892

2020-08-20 10:09:00.774 ru.netology.nmedia I/System.out: Привет, Мир!

CONTEXT

- в XML — `@string/hello`
- в коде — `getString(R.string.hello)`

Q: почему в XML достаточно просто указать имя, а в коде — нет?

A: при работе из кода вы должны использовать либо:

- helper-функции, которые по id ресурса получают сам ресурс (для XML это происходит автоматически;
- API, которое по id ресурса самостоятельно извлекает ресурс.
Нужно либо передавать контекст, либо у объекта уже есть ссылка на контекст.

CONTEXT

Самое важное, что нужно запомнить: эти helper-функции доступны только в классах-наследниках **Context**:

AppCompatActivity

Kotlin | **Java** ← смотреть именно в режиме Java

```
public class AppCompatActivity  
extends FragmentActivity implements AppCompatActivity,  
TaskStackBuilder.SupportParentable,  
ActionBarDrawerToggle.DelegateProvider
```

```
java.lang.Object
```

```
→ ↳ android.content.Context  
    ↳ android.content.ContextWrapper  
        ↳ android.view.ContextThemeWrapper  
            ↳ android.app.Activity  
                ↳ androidx.activity.ComponentActivity  
                    ↳ androidx.fragment.app.FragmentActivity  
                        ↳ androidx.appcompat.app.AppCompatActivity
```



CONTEXT

[Context](#) позволяет получать доступ к окружению приложения:
ресурсам и операциям по взаимодействию с ОС.

ЯЗЫК И СТРОКИ

Начиная с сегодняшней лекции, мы будем требовать вынесения всех строк, используемых в интерфейсе (далее — UI), в strings, как минимум, в двух вариантах:

1. Default (английский),
2. Configuration Specific (русский).

Важно:

1. Default обязательно должны быть (по крайней мере для строк).
2. Если в default не будет какого-то значения, а в configurations specific будет, то при конфигурации, не соответствующей configurations specific приложение будет остановлено с ошибкой.

ЛОКАЛИЗАЦИЯ

Вынесение строк UI в ресурсы позволяет:

- обеспечить соответствие UI текущей конфигурации устройства;
- избежать дублирования: одну и ту же строку можно использовать в разных элементах интерфейса;
- осуществлять перевод: при работе с китайским языком, вы не будете переводить сами, а отдадите переводчику;
- осуществлять изменения: достаточно заменить перевод в файле строк (код не изменится).



ЛОКАЛИЗАЦИЯ

Локализовать можно не только строки: любые ресурсы (изображения, layout и т.д.).



QUALIFIERS

QUALIFIERS

Что ещё можно использовать кроме языка:

- ширину и высоту экрана (w<N>dp, h<N>dp);
- **ориентацию экрана**: port, land;
- режим: night, notnight;
- **плотность пикселей**: mdpi, hdpi, xhdpi, xxhdpi, xxxhdpi, anydpi и т.д.;
- **версии платформы**: v4, v8, и т.д.;
- другие ([см. полный список](#)).

Qualifiers можно группировать:  ic_launcher.xml (anydpi-v26)

-
- наиболее используемые

SCREEN DENSITY

Если с версией платформы и ориентацией экрана всё понятно (некоторые возможности доступны только с определённой версии), то с плотностью пикселей нужно разобраться детально.

У разных экранов — разная разрешающая способность (плотность пикселей на дюйм экрана). Поэтому одна и та же картинка размером 100x100 пикселей будет выглядеть по-разному на разных экранах:



одна и та же картинка на разных экранах: у нижнего плотность в 4 раза больше, чем у верхнего

SCREEN DENSITY

Это большая проблема. Чтобы её решить, ввели специальные единицы измерений:

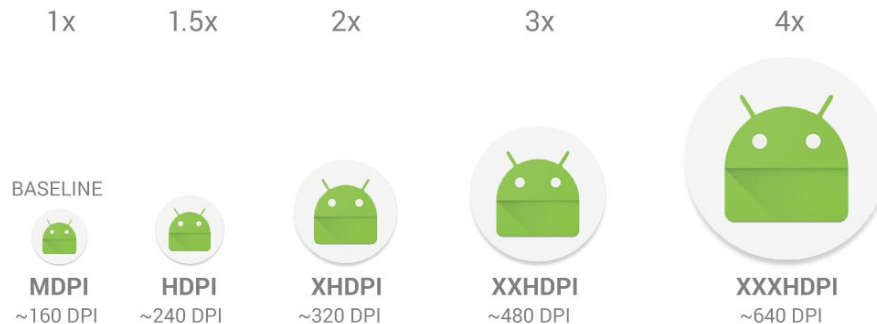
- **dp** (density independent pixels) — специальная единица измерения в UI, которая представляет из себя условных пиксель, масштабируемый в зависимости от плотности экрана.
- **sp** (scale independent pixels) — специальная единица для шрифтов, такая же как dp, но при этом учитывает настройки масштабирования шрифта, выбранные пользователем.

не путать с dp

1 dp = 1 px на экране с плотностью 160 точек на дюйм (dots-per-inch - dpi).

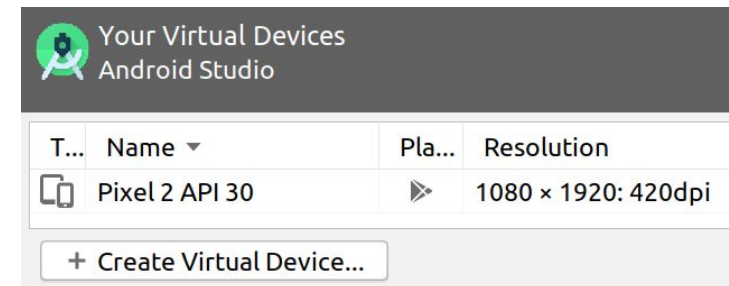
Основная идея: одинаковые физические размеры (в миллиметрах или дюймах) на разных устройствах. Например, «касание» (fingertip) — 50 dp.

SCREEN DENSITY



$$px = dp * (dpi / 160)$$

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
        println(resources.displayMetrics.heightPixels) // 1794  
        println(resources.displayMetrics.widthPixels) // 1080  
        println(resources.displayMetrics.densityDpi) // 420  
        println(resources.displayMetrics.density) // 2.625  
    }  
}
```



Ключевое: все размеры в dp, размеры текста в sp.

QUALIFIERS

Поэтому для растровых изображений (png, jpg) необходимо иметь копии как минимум для: mdpi, hdpi, xhdpi, xxhdpi, xxxhdpi.

Q: получается на одно изображение в приложении, мне нужно создавать 5 копий разного размера?

A: лучше пользоваться инструментами для дизайна, в которые это уже встроено (например, Figma с соответствующими плагинами умеет сразу экспортировать все ресурсы в нужных разрешениях).



[Figma](#) — сервис для разработки интерфейсов.



QUALIFIERS

Другой вариант: можно использовать векторную графику, которая автоматически масштабируется. Но для фотографий и сложных рисунков придётся хранить несколько копий, либо настраивать их ресайзинг (изменение размера) при компиляции.

SVG

[SVG](#) (Scalable Vector Graphics) — язык, основанный на XML, позволяющий описывать двухмерную векторную графику (можно встраивать и растровую).

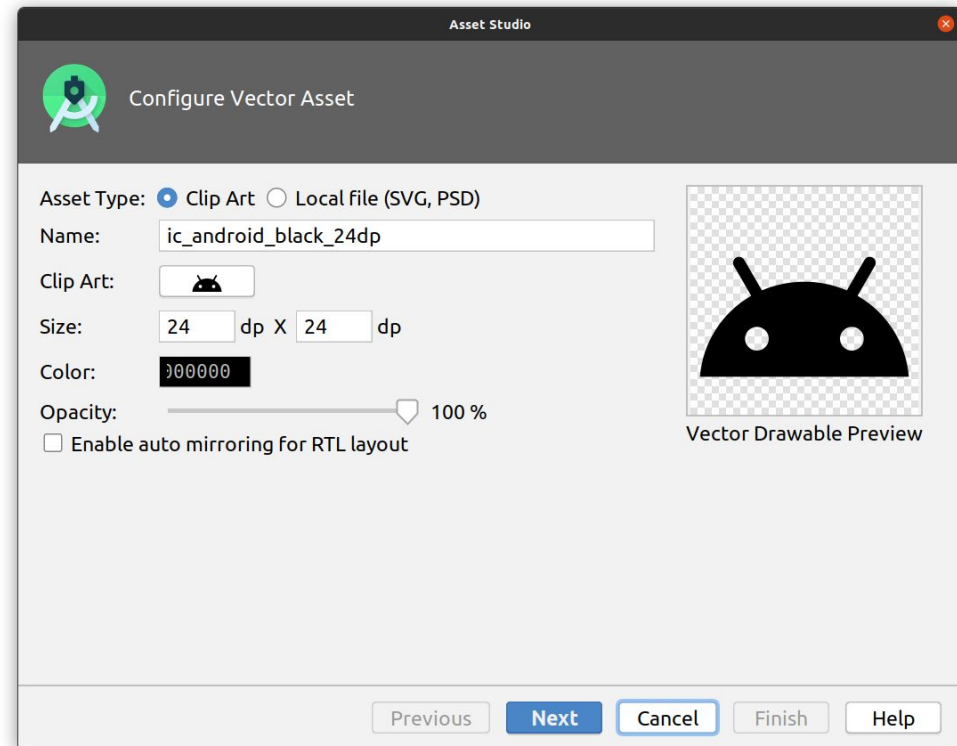
Векторную означает, что «фигуры» в изображении представлены не пикселями, а формулами, которые вычисляются и по ним уже строится изображение на конкретном устройстве (это и есть масштабируемость).

```
<svg width="460" height="460" viewBox="0 0 460 460" fill="none" xmlns="http://www.w3.org/2000/svg">
  <circle r="109.5" transform="matrix(-1 0 0 1 350.5 350.5)" fill="#0066FF" />
  <circle cx="109.5" cy="350.5" r="109.5" fill="#5D00F5"/>
  <circle cx="109.5" cy="109.5" r="109.5" fill="#EB236B"/>
  <circle cx="350.5" cy="109.5" r="109.5" fill="#4BD0A0"/>
  <ellipse cx="227.5" cy="229" rx="109.5" ry="110" fill="white"/>
</svg>
```

VECTOR DRAWABLE

SVG нельзя напрямую использовать в Android, но можно преобразовать в Vector Drawable (формат, поддерживаемый Android).

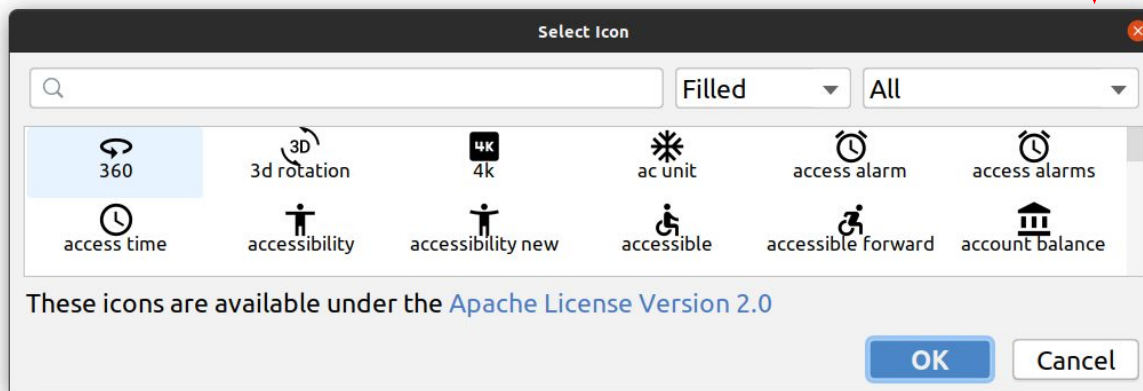
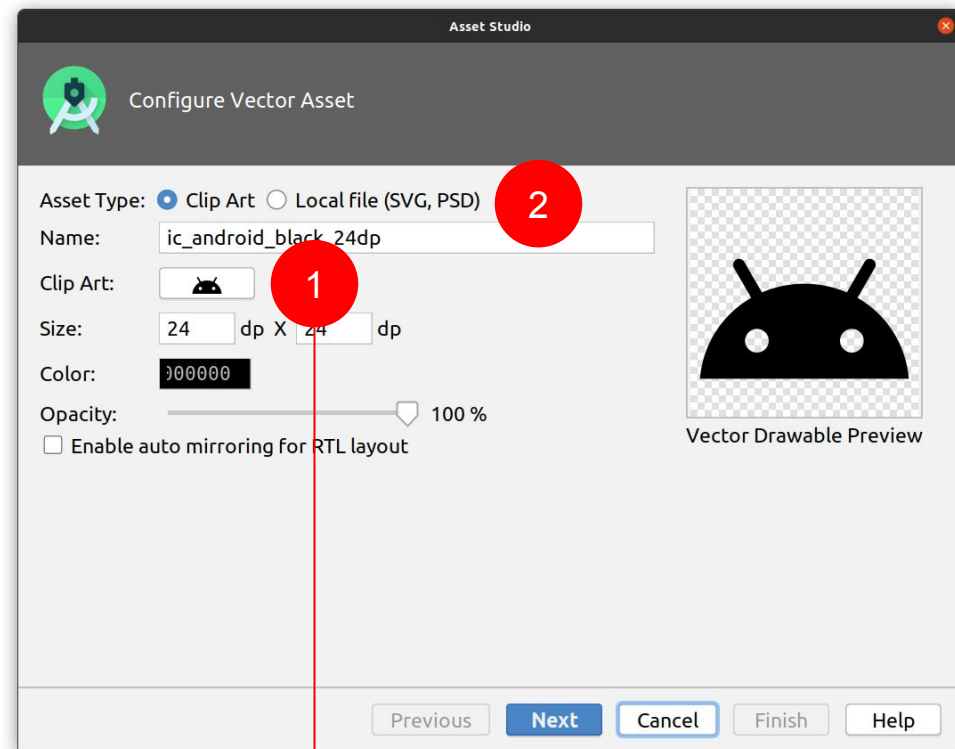
Для этого есть специальный инструмент Vector Asset Studio (New → Vector Asset на каталоге drawable):



VECTOR DRAWABLE

Есть две опции:

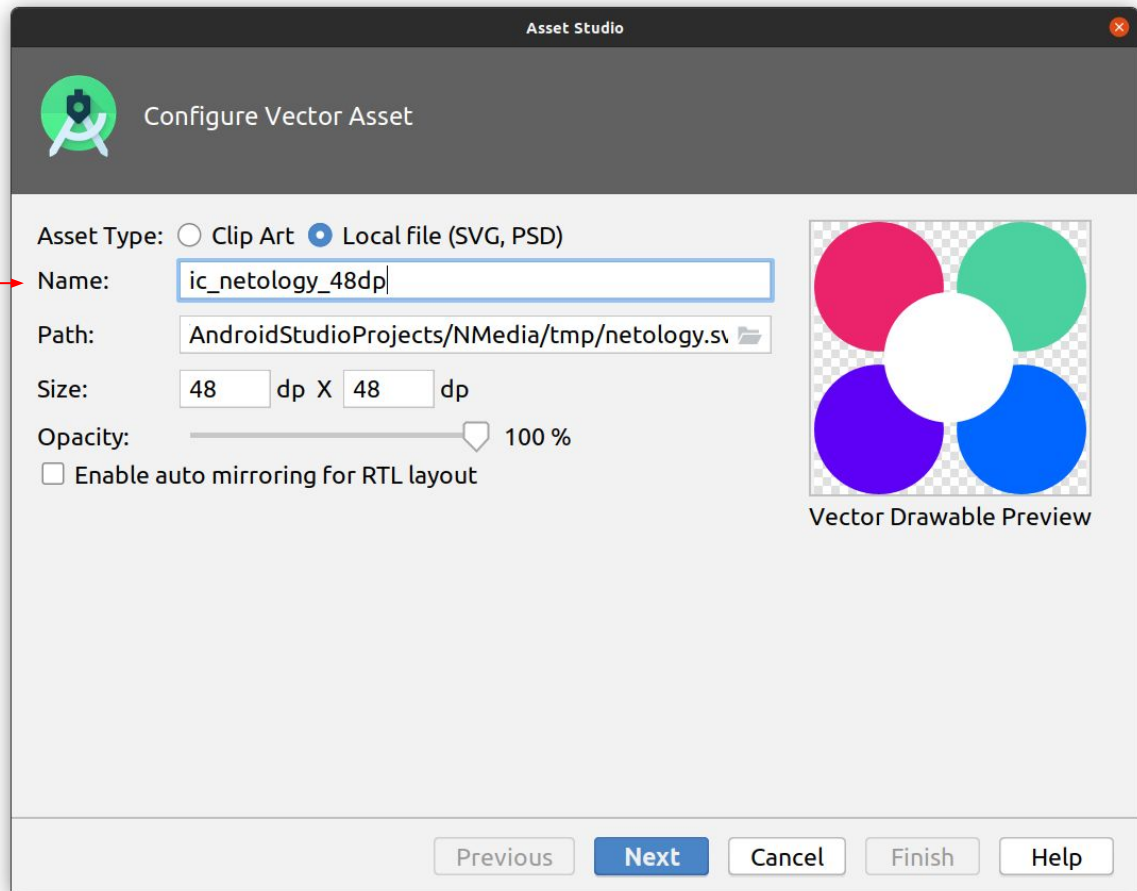
1. Использовать готовый Clip Art (1).
2. Импортировать локальный файл (SVG, PSD) (2).



Набор стандартных clip art'ов достаточно богат, вы можете использовать его во время обучения.

VECTOR DRAWABLE

Принято указывать в
названии размер



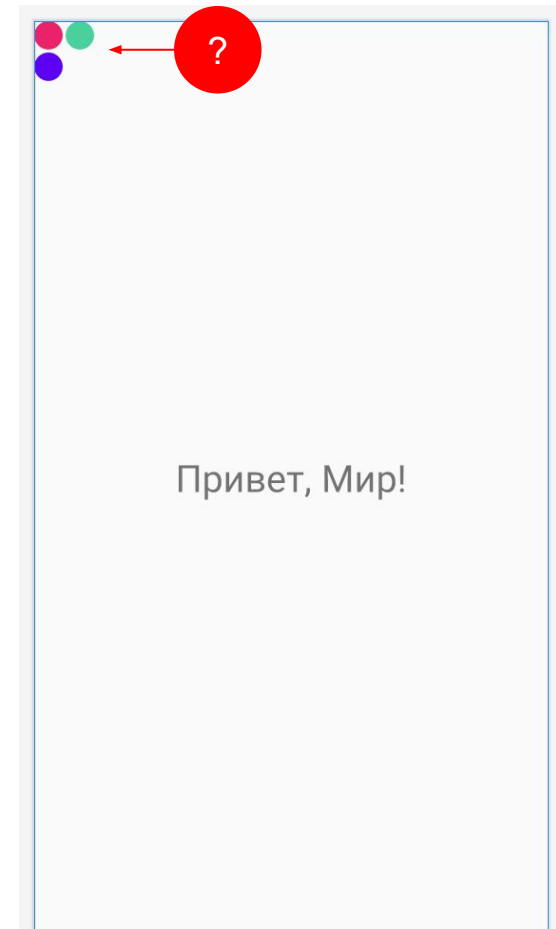
VECTOR DRAWABLE

После этого мы можем разместить в нашем XML компонент
ImageView:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    android:textSize="30sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<ImageView
    android:id="@+id/imageView"
    android:src="@drawable/ic_netology_original_48dp"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    />
```



Preview



И тут целых две проблемы:

1. картинка «битая»,
2. элемент улетел наверх.

VECTOR DRAWABLE

Важно: не все возможности SVG поддерживаются Android. Вполне возможна ситуация, когда после импорта вместо  получите 

Причём во время работы на самом устройстве оно **может** отображаться нормально, а вот в визуальном редакторе — нет.

Причина в том, что дизайнер неправильно выгрузил файл и нужно его попросить перевыгрузить. Figma и аналогичные инструменты обычно выгружают правильно и с ними таких проблем нет.

Вы можете это посмотреть на примере файлов [netology_original.svg](#) и [netology.svg](#) (в каталоге [import](#) демо-проекта).



SVG

Vector Drawable поддерживается с версии 20+ (Android 4.4+). В более ранних версиях необходимо либо настроить генерацию PNG из Vector Drawable, либо использовать специальную библиотеку Support Library (сейчас входит в состав AndroidX, которая у нас уже подключена).

—
UI



UI

Чтобы разобраться с тем, почему компонент улетел вверх, нам нужно в целом поговорить о UI и способах компоновке — взаимного размещения компонент на экране.

VIEW

Всё, что есть на экране, представлено наследниками класса View:

TextView

[Kotlin](#) | **Java**

```
public class TextView
extends View implements ViewTreeObserver.OnPreDrawListener
```

```
java.lang.Object
↳ android.view.View
↳ android.widget.TextView
```

ConstraintLayout

[Kotlin](#) | **Java**

```
public class ConstraintLayout
extends ViewGroup
```

ошибка в документации

```
java.lang.Object
↳ ViewGroup
↳ androidx.constraintlayout.widget.ConstraintLayout
```

ViewGroup

[Kotlin](#) | **Java**

```
public abstract class ViewGroup
extends View implements ViewParent, ViewManager
```

```
java.lang.Object
↳ android.view.View
↳ android.view.ViewGroup
```

VIEW & VIEWGROUP

[View](#) представляет базовый блок для UI — **прямоугольную область**, которая отвечает за свою отрисовку (какие нарисовать пиксели на экране) и обработку событий (как реагировать на касания и т.д.)

[ViewGroup](#) — базовый класс для layout'ов (не путать с ресурсами). Layout'ы — специальные элементы, управляющие размещением других элементов (позиционирование, размеры и т.д.). Условно их можно назвать «контейнерами» для других элементов (включая другие контейнеры).

HIERARCHY

Весь набор графических компонентов организован в иерархическое дерево, где у каждого компонента может быть ровно один родительский/ При этом у родителя может быть произвольное количество детей.

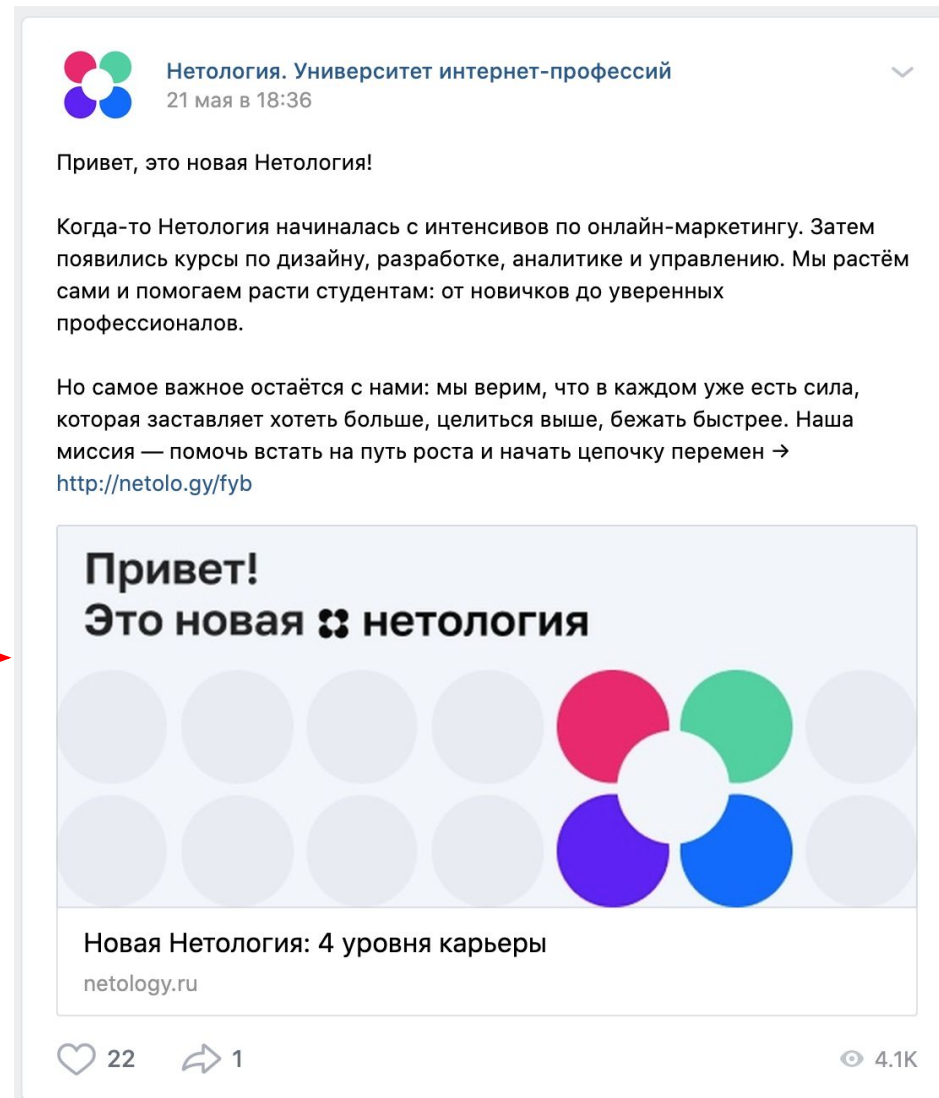
Исходя из этого становится возможным определять размеры компонента:

- 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 (например, всё свободное пространство от границы одного компонента до границы другого).

LAYOUT

После того, как мы разобрались с ресурсами, следующим шагом станет изучение возможностей компонентов и их компоновки.

В частности, на следующей лекции мы с вами решим задачу отображения карточки поста на экране:





ИТОГИ

ИТОГИ

Сегодня мы обсудили с вами вопросы, связанные с ресурсами:

- default и configuration specific;
- локализацию (в первую очередь строк);
- особенности работы с изображениями.