

Изображения

Все что нужно знать про картинки

Содержание

1. Изображения статичные
2. Получаем из сети

1. Изображения статичные

В предыдущей лекции мы рассмотрели текстовки. В конце лекции мы добавили изображение к тексту. Но что если нам не нужен текст вовсе? Не писать же пустой текст и картинку к нему? Нет, конечно же. Для этого есть другая вью – `ImageView`. Давайте заменим `TextView` на `ImageView` в нашем `activity_main` лейауте.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <ImageView
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     android:layout_width="wrap_content"
5     android:layout_height="wrap_content"
6     android:src="@android:drawable/ic_menu_camera" />
```

Для того чтобы наша вью показывала хоть что-то нужно написать атрибут `src` что означает source и указать путь к drawable. У нас в наших ресурсах особо ничего и нет сейчас, поэтому мы воспользуемся опять же ресурсами из андроид - `@android:drawable/`. Но как видите у нас вылезло предупреждение в правом верхнем углу и тег `ImageView` подсвечен желтым. Что же не так?

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <ImageView
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     android:layout_width="wrap_content"
5     android:layout_height="wrap_content"
6     android:src="@android:drawable/ic_menu_camera" />
```

ImageView

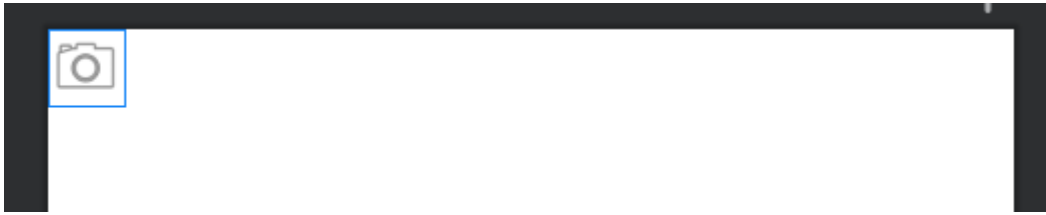
Problems: Current File 1

activity_main.xml ~/AndroidStudioProjects/HelloWorld/app/src/main/res/layout 1 problem

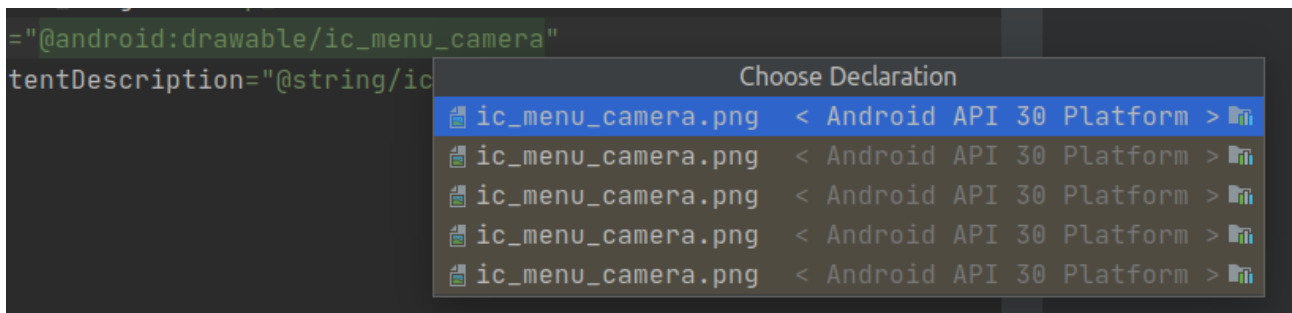
Missing `contentDescription` attribute on image :2

Нам необходимо предоставить описание содержимого для слабовидящих – есть такая штука как `TalkBack` и она проговаривает контент на экране, когда видит изображение, то как

объяснить что на нем? Через строку которая указана в `contentDescription`. Вы можете забыть на предупреждение и заигнорировать, или же написать описание и положить в строковые ресурсы. Решать вам. Посмотрим же как выглядит дизайн экрана

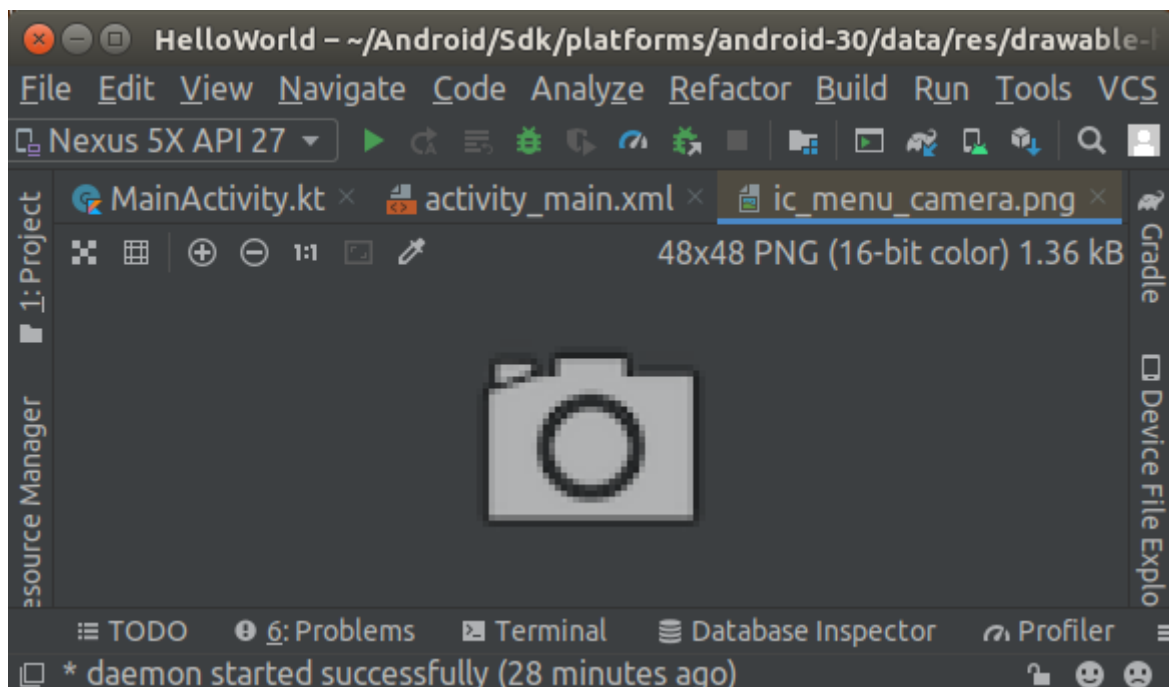


У нас ширина и высота опять стоят по содержимому – `wrap_content` и потому наша картинка не растянута ни по ширине ни по высоте. Давайте провалимся в ресурс и посмотрим что там лежит



Как видите мы имеем 5 png файлов. Почему 5? И в чем их отличие? Давайте посмотрим

Нажмем на первое



Мы видим нашу иконку и ее размер в правом верхнем углу – 48x48 – да, это пиксели.

Аналогично нажмем на второй файл и посмотрим что там – а там 36x36. Ок, а дальше?

32x32, 64x64 и 96x96. Что значат эти числа и что между ними общего? И здесь мы впервые сталкиваемся с тем, что Android устройства имеют разное разрешение и плотность экрана.

Предположим у вас обычный смартфон с разрешением Full HD – 1920×1080.

А это значит 1920 пикселей по оси ординат (по вертикали) и 1080 пикселей по оси абсцисс (по горизонтали). На самом деле легко заметить что разрешение у нас 640х360 но помноженное на 3. А значит если мы хотим чтобы изображение со стороной 48 выглядело одинаково везде, то для нашего устройства с Full HD нам придется использовать 48*3 = 144.

Именно поэтому у нас 5 png файлов. Давайте посмотрим что говорит официальная документация по этому поводу (ldpi в 2021 году уже нельзя найти, так что забейте на это).

- 36x36 (0.75x) for low-density (ldpi)
- 48x48 (1.0x baseline) for medium-density (mdpi)
- 72x72 (1.5x) for high-density (hdpi)
- 96x96 (2.0x) for extra-high-density (xhdpi)
- 144x144 (3.0x) for extra-extra-high-density (xxhdpi)
- 192x192 (4.0x) for extra-extra-extra-high-density (xxxhdpi)

Дизайнеры верстают экраны для mdpi для средней плотности. Но чтобы вам не нужно было думать над тем сколько должен быть размер для других плотностей и разрешений он вам предоставит пакет из 5 png файлов. Но точно так же как мы создавали второй strings.xml для русской локали мы должны создать 5 пакетов под картинки чтобы не писать в коде when.

```
res/  
  drawable-xxxhdpi/  
    awesome-image.png  
  drawable-xxhdpi/  
    awesome-image.png  
  drawable-xhdpi/  
    awesome-image.png  
  drawable-hdpi/  
    awesome-image.png  
  drawable-mdpi/  
    awesome-image.png
```

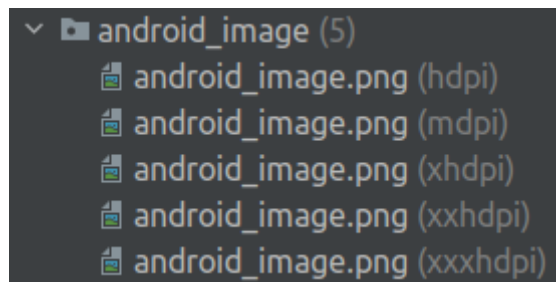
И положить в соответствующий пакет каждый файл.

И кто-то сейчас скажет – реально так сложно просто показывать картинки? Ну да. Хотя на самом деле нужно понимать что такое растровое изображение и векторное. Кроме png есть еще svg векторные изображения. Их суть в том, что они не нарисованы в paint (условно говоря) и хранят в себе формулы по которым можно нарисовать на экране векторное изображение и вам уже все равно какая плотность экрана и т.д. Чтобы понимать лучше давайте рассмотрим детальнее. Для начала попробуем скачать png иконку, точнее 5 штук и положить в соответствующие пакеты. Время погуглить!

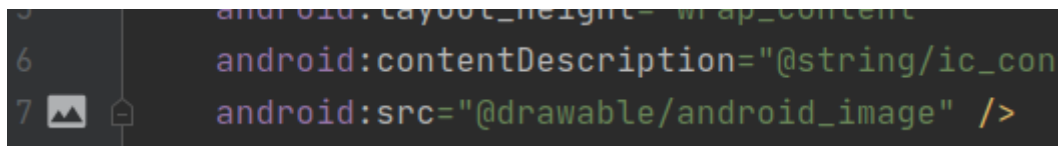
Я нашел png файл с разрешением 768 на 768. Значит мне нужно разделить на 4 и я получаю 192 для базовой версии. Далее мне нужно переделать их размеры по логике 1x, 1.5x, 2x, 3x а для 4x файл оригинал добавить в пакет drawable-xxxhdpi. Ну и да, нужно руками создать пакеты – для этого нажимаем правый клик и идем в место где находится файл в пакете drawable.



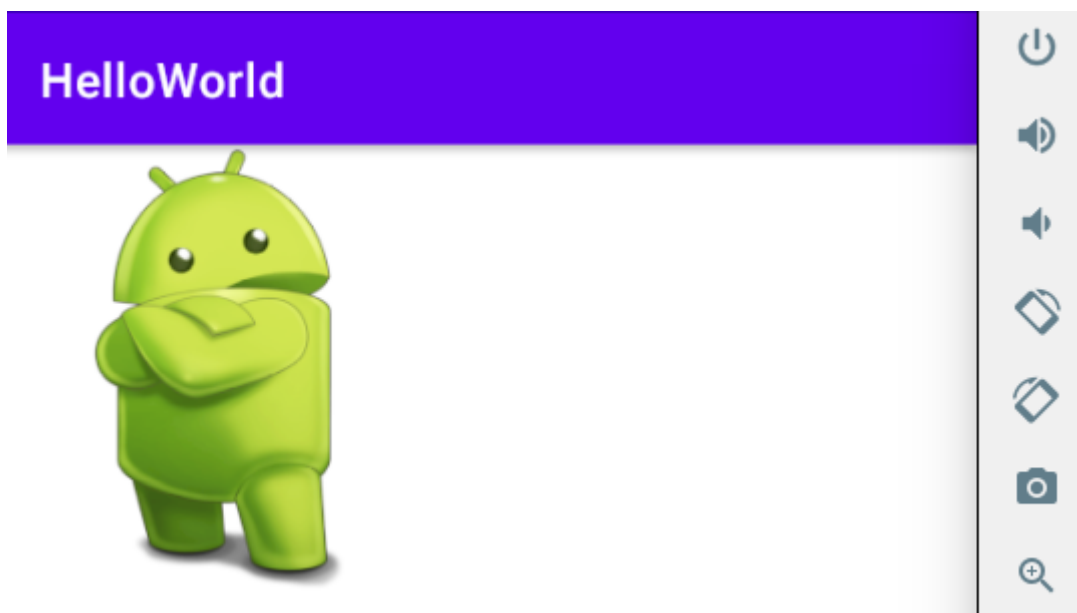
И после добавления 5 файлов вернемся в АС и посмотрим что вышло



В виде дерева Android у нас будет так – пакет под изображение в котором все 5 файлов. На самом деле у нас 5 пакетов с 5 файлами внутри и все названия файлов одинаковые. Ладно, как же теперь использовать наш png которых 5 штук? Смотрим



Можем запустить проект и увидим следующее



Вроде как все правильно отображается, да? Как это проверить?

Давайте напишем размеры руками – раньше у нас было `wrap_content`, а сейчас напишем `192dp` и вынесем в ресурсы

```
android:layout_width="@dimen/image_size"
android:layout_height="@dimen/image_size"
```

Теперь уж точно все должно быть верно и не нужно проверять на разных устройствах. Хотя вы можете это сделать прямо в АС. Там где у нас превью можете выбрать различные устройства с различными разрешениями и проверить отображение. В чем суть этих 5 файлов? Чтобы на всех устройствах все выглядело более менее одинаково – т.е. если вы зададите высоту и ширину в пикселях (да, так можно), то на одном устройстве изображение будет занимать условно половину экрана по ширине, а на другом устройстве лишь четверть.

И давайте еще раз проговорим – если ваше изображение было создано такими инструментами что в итоге оно растровое – вам нужно 5 файлов png. Если же ваш файл был создан векторным, то здесь все намного проще. Идем в гугл и скачиваем svg файл для иконки. <https://fonts.google.com/icons?selected=Material+Icons> Здесь можно скачать архивом как все 5 пнг файлов так и сразу вектор и можно положить в проект готовое. Но я вам хочу показать как нужно работать с svg

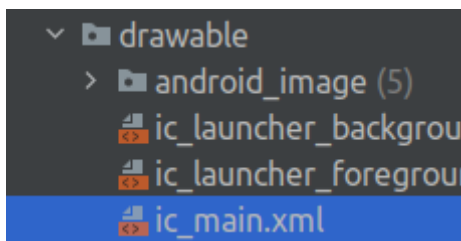
```
1 <?xml version="1.0" encoding="UTF-8"?><!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd"><svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1" width="24" height="24" viewBox="0 0 24 24"><path d="M4,6H2V20A2,2 0 0,1 22,4V16A2,2 0 0,1 20,18H8A2,2 0 0,1 6,16V4A2,2 0 0,1 8,2H20M17,7A3,3 0 0,0 14,4A3,3 0 0,0 11,7A3,3 0 0,0 14,10A3,3 0 0,0 17,7M8,15V16H20V15C20,13 16,11.9 14,11.9C12,11.9 8,13 8,15Z" /></svg>
```

Если открыть svg файл в блокноте, то вы увидите нечто подобное. Нас это не устраивает и потому нам нужно конвертировать в вектор для андройд. Для этого есть сайт <http://inloop.github.io/svg2android/> с его помощью получаем нужный вектор

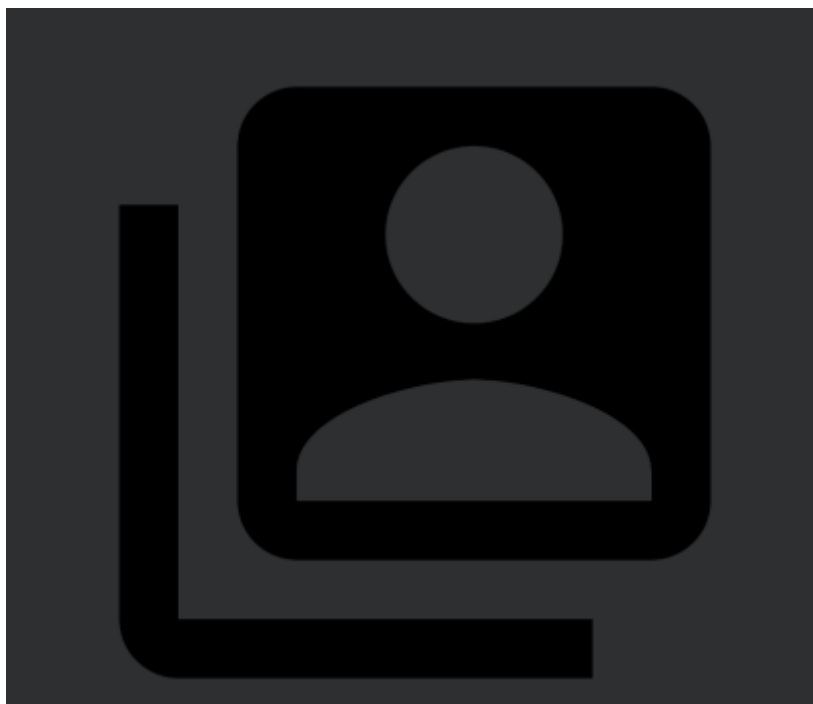
```
<?xml version="1.0" encoding="utf-8"?>
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="24dp"
    android:height="24dp"
    android:viewportWidth="24"
    android:viewportHeight="24">

    <path
        android:fillColor="#000000"
        android:pathData="M4,6H2V20A2,2 0 0,1 22,4V16A2,2 0 0,1 20,18H8A2,2 0 0,1 6,16V4A2,2 0 0,1 8,2H20M17,7A3,3 0 0,0 14,4A3,3 0 0,0 11,7A3,3 0 0,0 14,10A3,3 0 0,0 17,7M8,15V16H20V15C20,13 16,11.9 14,11.9C12,11.9 8,13 8,15Z" />
</vector>
```

Копируем все это дело и создаем в пакете `drawable` новый файл типа `ic_main` для использования в `imageView`.



Теперь посмотрите на превью картинки -



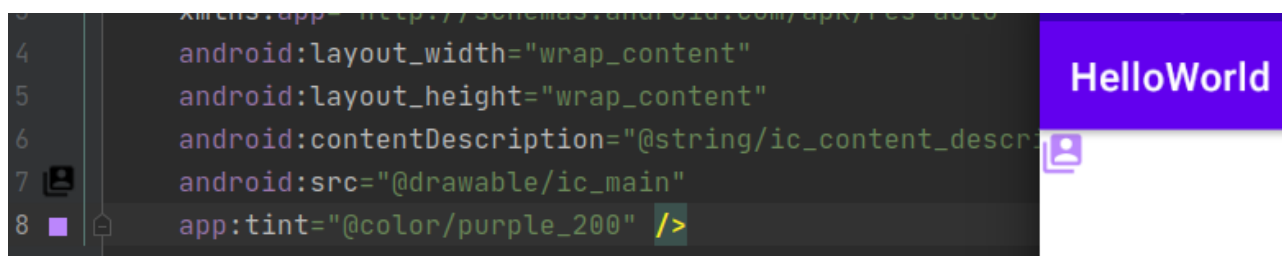
В хмл вернем `wrap_content` для высоты и ширины, потому что в векторе уже прописаны высота и ширина в `24dp`. Кстати, если вы захотите поменять цвет, то это просто -

```
<path
    android:fillColor="#000000"
    android:pathData="M4,6H2V20A2
```

Здесь меняем цвет и все – уже иконка иного цвета. Давайте уже запустим проект



Мой вам совет – при возможности используйте векторы нежели png. Никому не охота импортировать в проект 5 файлов. 1 всяко легче и проще. И да, если вам надо все равно поменять цвет то у вас есть tint



Вы всегда можете прочитать о всех атрибутах по ссылке на офф.документацию <https://developer.android.com/reference/android/widget/ImageView> также про методы класса. Ладно. Это все понятно. Что же с изменением из кода? Все просто (если у нас константы).

```
2 <ImageView xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     android:layout_width="wrap_content"
5     android:layout_height="wrap_content"
6     android:id="@+id/iconImageView"
7     android:contentDescription="@string/ic_content_description"
8     android:src="@drawable/ic_main"
9     app:tint="@color/purple_200" />
```

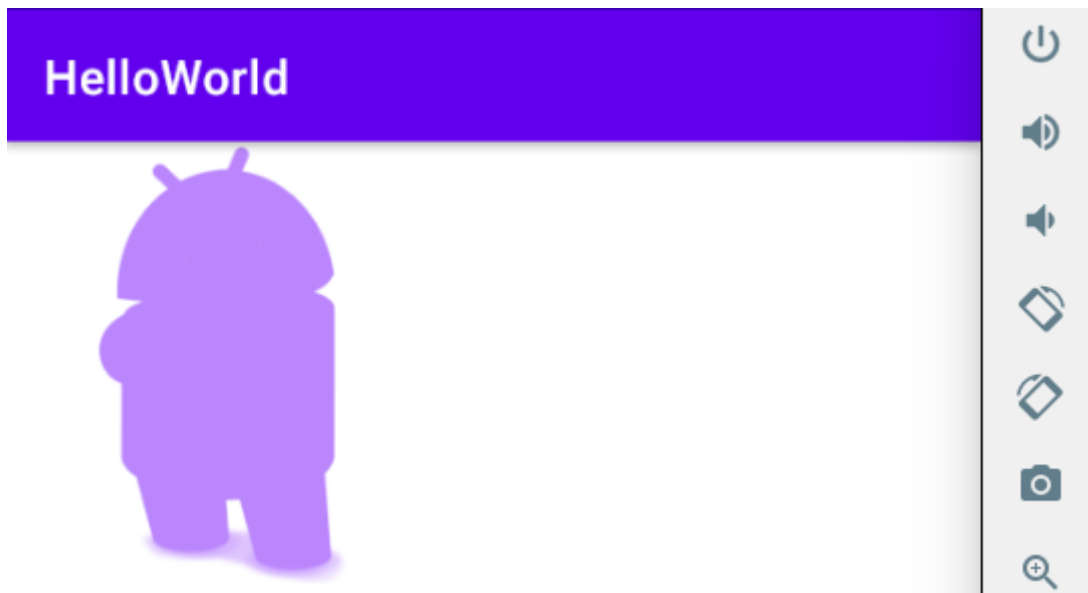
Добавляем айди для того чтобы из кода можно было получить доступ к вью.

```
val image = findViewById<ImageView>(R.id.iconImageView)
image.setImageResource(R.drawable.android_image)
```

В коде просто сетим иной ресурс – заметьте что у нас метод принимает опять же int но с аннотацией @DrawableRes чтобы различать от других (пример из первой лекции @StringRes)

```
public void setImageResource( @DrawableRes @DrawableRes int resId) {
```

Запустим код и увидим – сначала в xml была иконка вектор фиолетового цвета и потом кодом поставили растровое png итого имеем вот это



Именно поэтому я и не рекомендую использовать растровые изображения – tint работает именно так – просто меняет цвет всего что видит, даже тени.

Кстати, точно так же как и с текстовкой вы можете поставить фоновый цвет.

2. Получаем из сети

Но все слишком просто когда речь идет об изображениях из ресурсов. Самое интересное начинается когда нам нужно скачать картинку из сети. Помните мы в джава части скачивали файл? Ровно такое же нужно написать и здесь. Давайте для начала поменяем нашу разметку таким образом чтобы картинка была во весь экран.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <ImageView xmlns:android="http://schemas.android.com/apk/res/android"
3     android:id="@+id/imageView"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6     android:contentDescription="@string/ic_content_description" />
```

Итак, правило первое – ваш файл активности не должен заниматься всем что можно особенно скачиванием файлов. Мы не зря учили ООП и потому напишем отдельный класс для скачивания изображения. Так же напишем колбек чтобы по готовности отобразить.

```
class NetImage(
    private val url: String,
    private val callback: ImageCallback
) : Thread() {

    override fun run() {
        super.run()
        try {
            val connection = URL(url).openConnection()
            connection.doInput = true
            connection.connect()
            connection.inputStream.use { it: InputStream!
                callback.success(BitmapFactory.decodeStream(it))
            }
        } catch (e: Exception) {
            callback.failed()
        }
    }
}
```

Не забываем закрывать стримы с помощью use и оборачиваем в try catch чтобы наше приложение не крашнулось (закончилось с ошибкой). Для этого у нас есть метод в колбеке с предоставлением сообщения об ошибке.

В колбеке мы в случае успеха отдаем битмар чтобы в активности уже установить ImageView.


```
interface ImageCallback {

    fun success(bitmap: Bitmap)

    fun failed()

}
```

И давайте в активности уже напишем поток и скачаем файл, посмотрим что выйдет

```
private companion object {
    const val URL =
        "https://zavistnik.com/wp-content/uploads/2020/03/Android-kursy-zastavka.jpg"
}

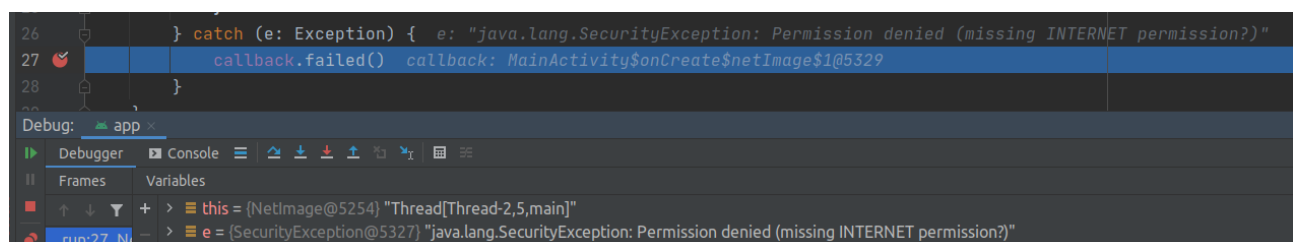
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    val image = findViewById<ImageView>(R.id.imageView)
    val netImage = NetImage(URL, object : ImageCallback {
        override fun success(bitmap: Bitmap) {
            image.setImageBitmap(bitmap)
        }

        override fun failed() {
            Snackbar.make(image, text: "failed", Snackbar.LENGTH_SHORT).show()
        }
    })
    netImage.start()
}
```

Создаем поток, пишем колбек анонимным и стартуем. Запустим проект!

Видим failed! Почему же? Подебажим! Ставим брейкпойнт на линии где у нас catch блок в классе NetImage



```
26 } catch (e: Exception) { e: "java.lang.SecurityException: Permission denied (missing INTERNET permission?)"
27     callback.failed() callback: MainActivity$onCreate$NetImage$1@5329
28 }
```

Debug: app

Frames

Variables

this = (NetImage@5254) "Thread[Thread-2,main]"

e = (SecurityException@5327) "java.lang.SecurityException: Permission denied (missing INTERNET permission?)"

У нас нет разрешения использовать интернет. Что? Да, в Андроид нельзя просто так взять и использовать интернет, нужно получить разрешение. Не беспокойтесь, это делается в 1 линию. Открываем Андроид Манифест и пишем там что приложению нужен интернет. Когда юзеры будут скачивать ваше приложение они увидят, что оно может использовать интернет соединение.

```
MainActivity.kt x NetImage.kt x AndroidManifest.xml x ImageView.java x activity_main.xml x
4
5     <uses-permission android:name="android.permission.INTERNET"/>
6     |
7     <application
```

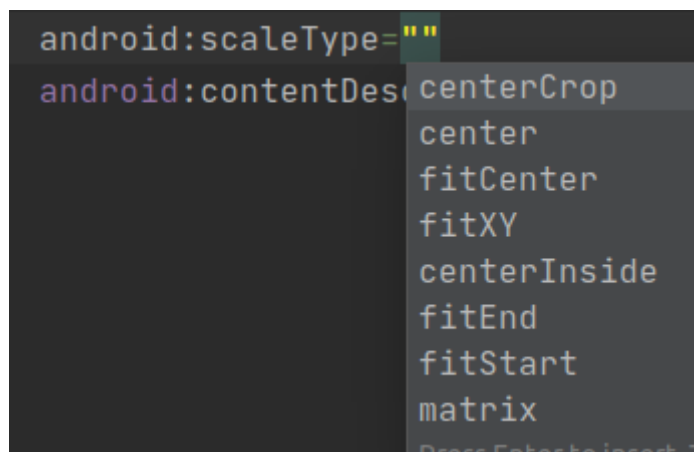
Начните писать `<uses` и АС предложит варианты. Подробнее про пермишены можете погуглить сами – все что требует пермишенов нужно декларировать. Там есть нюанс. Но об этом в другой лекции. А пока давайте запустим еще раз наш проект и посмотрим работает ли



Все скачалось! Неожиданно! Но мы же написали чтобы картинка была по всей ширине и по всей высоте, почему оно по центру экрана? Потому что у картинки ширина больше высоты и получается что для портретного режима не подходит. Ок, может повернем эмулятор?

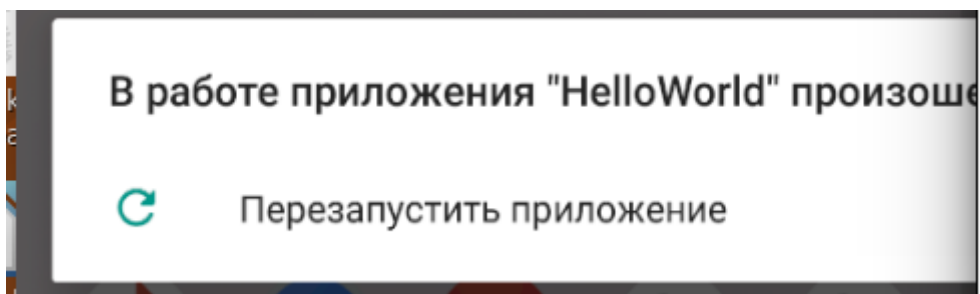


Как видите даже в этом случае не все пространство заполнено. Почему же так? А потому что ширина и высота не совпали с параметрами экрана. Как же пофиксить это?

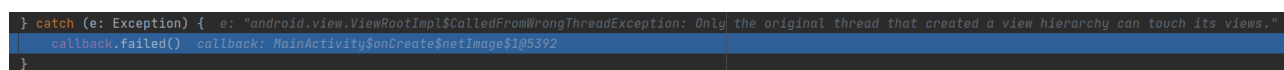


Итак, время познакомиться с атрибутом `scaleType` – как нам масштабировать картинку – есть много разных вариантов и я предлагаю вам самим выбирать по очереди каждый и понять что происходит в каждом случае. А для большей ясности давайте повернем обратно девайс.

Для примера скажу что я использовал `fitXY` и изображение растянулось на весь экран в портретном режиме. Это конечно же фу, но что поделать. Вы должны понимать что с сервера может прийти абсолютно любое изображение в любом качестве, с любым соотношением сторон и так далее. Кстати, а что с нереально большими файлами? Давайте возьмем изображение неимоверно большое, сможет ли андроид с этим справиться? (спойлер, нет).



Крашнулось! Погуглите 8к wallpaper с разрешением 9000*5000



Only the original thread that created a view hierarchy can touch its views.

Только оригинальный поток, который создал вью иерархию может трогать вьюшки. Что?

На самом деле я ожидал эту ошибку еще в первый раз когда мы скачивали первую картинку.

Помните я говорил что если мы запускаем новый тред в текущем, то мы получаем уже 2 треда. В Андроид тот тред, в котором работает `ui` называется `main`-тред или `ui-thread` что одно и то же. Так вот – в Андроид нельзя делать сложные вещи на главном потоке (качать файлы – будет лагать юай), кроме этого нельзя данные полученные из другого потока сетить (`set`) в вью главного потока. Как же это решается? В Активити есть метод `runOnUiThread` – посмотрите как это работает

В котлин можно просто добавить после методов это и будет работать

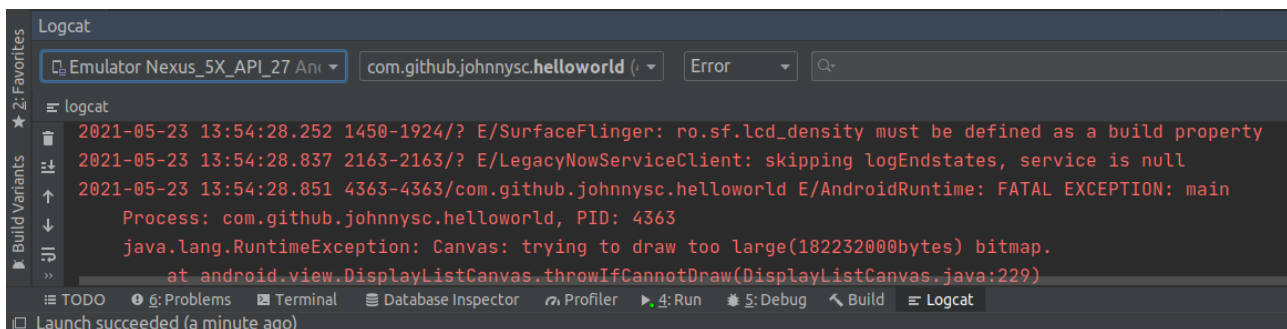
```

override fun success(bitmap: Bitmap) = runOnUiThread {
    image.setImageBitmap(bitmap)
}

override fun failed() = runOnUiThread {
    Snackbar.make(image, text: "failed", Snackbar.LENGTH_SHORT).show()
}

```

Теперь уж все должно быть ок, да? Нет. Посмотрите вниз в АС. Есть Logcat и там можно увидеть что пошло не так – выбираем фильтр Error и можно выбрать девайс и процесс.



Logcat

Emulator Nexus_5X_API_27 Android

com.github.johnnysc.helloworld

Error

logcat

2021-05-23 13:54:28.252 1450-1924/? E/SurfaceFlinger: ro.sf.lcd_density must be defined as a build property

2021-05-23 13:54:28.837 2163-2163/? E/LegacyNowServiceClient: skipping logEndstates, service is null

2021-05-23 13:54:28.851 4363-4363/com.github.johnnysc.helloworld E/AndroidRuntime: FATAL EXCEPTION: main

Process: com.github.johnnysc.helloworld, PID: 4363

java.lang.RuntimeException: Canvas: trying to draw too large(182232000bytes) bitmap.

at android.view.DisplayListCanvas.throwIfCannotDraw(DisplayListCanvas.java:229)

Launch succeeded (a minute ago)

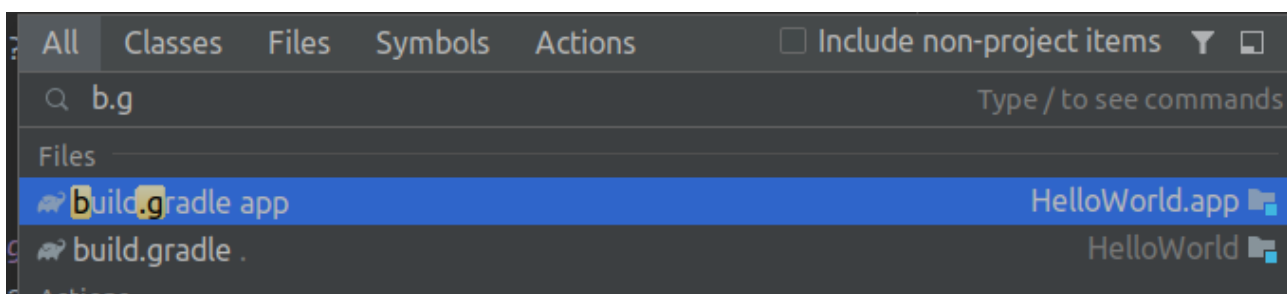
Как видите FATAL EXCEPTION – trying to draw too large bitmap. Чаво? Ага, слишком большой файл для мобилки. Вот так вот. Что же делать? Нужно как-то менять размеры файла во время получения из сети.

А еще было бы неплохо чтобы приложение не крашилось, а показывало бы ошибку прямо в том же месте где должно быть изображение. Еще бы было классно чтобы была загрузка пока юзер ждет скачивания изображения. Плюс можно было бы кешировать скачанное чтобы каждый раз не скачивать. А еще нам возможно захочется преобразовать в круг – ну как в аватарках например. Если мы начнем писать код под все эти требования, то не закончим до нового года. Да и писать Thread слишком затратно для каждого асинхронного процесса. Нет, речь сейчас не пойдет про корутины. Речь про готовые библиотеки.

Проблемы загрузки, кеширования и преобразования картинок были решены более крутыми разработчиками давным давно и нам доступны готовые библиотеки. Можно просто взять и использовать их. Для примера давайте рассмотрим популярную библиотеку Picasso

<https://square.github.io/picasso/> можете перейти на офф.сайт либы (библиотеки lib) и глянуть

Чтобы иметь доступ к этой либе нужно ее подключить через Gradle. Для этого в АС делаем 2 раза Shift Shift и пишем b.g и выбираем build.gradle (да, пишете 2 буквы файла и разрешения и АС предложит).



All Classes Files Symbols Actions

Include non-project items

b.g

Type / to see commands

Files

build.gradle app HelloWorld.app

build.gradle HelloWorld

Actions

Выбираем первое и добавляем зависимость (подробнее про градл поговорим в иной лекции)

```
35 dependencies {
36     implementation 'com.squareup.picasso:picasso:2.71828'
37     implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
38     implementation 'androidx.core:core-ktx:1.3.2'
```

Добавили в проект либу пикассо и надо скачать ее исходники чтобы можно было использовать. Для этого наверху жмем Sync Now вы увидите как только вставите имя либы.

```
val image = findViewById<ImageView>(R.id.imageView)

Picasso.get().load(URL).centerCrop()
    .resize(targetWidth: 720, targetHeight: 1280)
    .placeholder(android.R.drawable.ic_media_pause)
    .error(android.R.drawable.ic_dialog_alert)
    .into(image)
```

Теперь у нас появился класс Picasso и можно загружать огромные картинки и по пути менять их размер. Здесь конечно же плохой код – 720 и 1280, но вы можете написать свою логику по какому принципу и до каких размеров нужно менять ширину и высоту. Далее мы пишем placeholder это то, что будет отображено пока скачивается картинка. .error для случая если не смогли скачать и конечно же into чтобы в итоге показать изображение в имеджвью. Как видите сразу же доступна centercrop и другие функции масштабирования. Кстати, теперь изображение закешировано и вам не надо ждать скачивания второй раз. Можете выйти из приложения и зайти снова. Если вам интересно что конкретно я скачивал, то вот линк.

```
const val URL = "https://images4.alphacoders.com/109/1095230.jpg"
```

Кстати, вы заметили что я по-быстрому заюзал андроид иконки через android.R.drawable ведь у нас был свой R класс и чтобы не было конфликта надо писать префикс андроид.

Но скажу вам – оказывается либа Picasso не так хорошо работает как хотелось бы и я пришел к тому что лучше юзать другую <https://github.com/bumptech/glide> – Глайд быстрее работает и проще для ресайза и т.д. Посмотрите как ее подключить в проект и замените пикассо на нее.

И да, вы всегда можете написать экстеншн функцию для того чтобы скрыть все детали внутри. Это не только уменьшит время написания кода, но и сделает проще проблему замены одной библиотеки на другую. Поверьте, если в вашем проекте 100 раз вызваны методы пикассо, то когда вы решите перейти на глайд то вам придется вносить изменения в 100 файлах. А если у вас экстеншн функции – то в 1 файле. Удобно!

И напоследок – атрибут adjustViewBounds очень часто используется чтобы картинка заполняла все пространство. Прочитайте документацию на офф.сайте чтобы понять суть.

```

21         val image = findViewById<ImageView>(R.id.imageView)
22         image.load(URL)
23     }
24 }
25
26 fun ImageView.load(url:String) {
27     Picasso.get()
28         .load(url)
29         .placeholder(android.R.drawable.ic_media_pause)
30         .error(android.R.drawable.ic_dialog_alert)
31         .into(target: this)
32 }

```

Как домашнее задание сначала сделайте загрузку круглого изображения на пикассо, после замените пикассо на глайд и сделайте там.