Практическая работа 6
Методические материалы
Ресурсы. Работа с изображениями.

Работа с ресурсами

Ресурс в приложении Android представляет собой файл, например, файл разметки интерфейса или некоторое значение, например, простую строку. То есть ресурсы представляют собой и файлы разметки, и отдельные строки, и звуковые файлы, файлы изображений и т.д. Все ресурсы находятся в проекте в каталоге **res**. Для различных типов ресурсов, определенных в проекте, в каталоге **res** создаются подкаталоги. Поддерживаемые подкаталоги:

- animator/: xml-файлы, определяющие анимацию свойств
- anim/: xml-файлы, определяющие tween-анимацию
- color/: xml-файлы, определяющие список цветов
- drawable/: Графические файлы (.png, .jpg, .gif)
- mipmap/: Графические файлы, используемые для иконок приложения под различные разрешения экранов
- layout/: xml-файлы, определяющие пользовательский интерфейс приложения

- menu/: xml-файлы, определяющие меню приложения

- raw/: различные файлы, которые сохраняются в исходном виде

- values/: xml-файлы, которые содержат различные используемые в приложении значения, например, ресурсы строк

- xml/: Произвольные xml-файлы

- font/: файлы с определениями шрифтом и расширениями .ttf, .otf или .ttc, либо файлы XML, который содержат элемент <font-family>

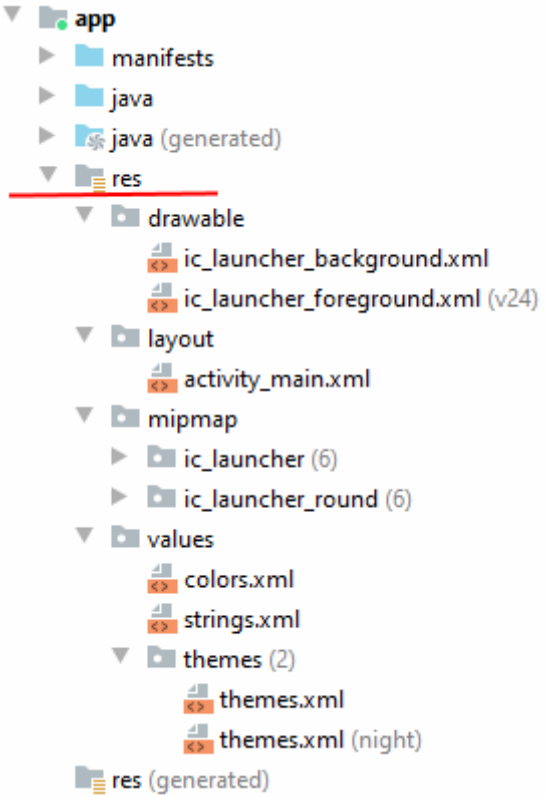
В общей сложности мы можем определить следующие типы ресурсов

Ресурс	Каталог проекта	Файл	элемент в файле
Строки	/res/values/	strings.xml	<string>
Plurals	/res/values/	strings.xml	<plurals>
Массивы строк	/res/values/	strings.xml или arrays.xml	<string-array>
Логические значения Boolean	/res/values/	bools.xml	<bool>
Цвета	/res/values/	colors.xml	<color>
Список цветов	/res/color/	Произвольное название	<selector>
Размеры (Dimensions)	/res/values/	dimens.xml	<dimen>
Идентификаторы ID	/res/values/	ids.xml	<item>
Целые числа	/res/values/	integers.xml	<integer>
Массив целых чисел	/res/values/	integers.xml	<integer-array>

Графические файлы	/res/drawable/	Файлы с расширением jpg и png	-
Tween-анимация	/res/anim/	Файл xml с произвольным названием	<set>, <alpha>, <rotate>, <scale>, <translate>
Покадровая анимация	/res/drawable/	Файл xml с произвольным названием	<animation-list>
Анимация свойств	/res/animator/	Файл xml с произвольным названием	<set>, <objectAnimator>, <valueAnimator>
Меню	/res/menu/	Файл xml с произвольным названием	<menu>
XML-файлы	/res/xml/	Файл xml с произвольным названием	
Бинарные и текстовые ресурсы	/res/raw/	Файлы мультимедиа (mp3, mp4), текстовые и другие файлы	

Разметка графического интерфейса	/res/layout/	Файл xml с произвольным названием	
Стили и темы	/res/values/	styles.xml, themes.xml	<style>

К примеру, если мы возьмем стандартный проект Android Studio, который создается по умолчанию, то там можем заметить наличие уже нескольких папок для различных ресурсов в каталоге res:



Ресурсы в Android Studio

По умолчанию здесь есть каталоги не для всех типов ресурсов, которые использоваться в Android, однако при необходимости мы можем добавить в папку res нужный каталог, а в него затем поместить ресурс.

Когда происходит компиляция проекта сведения обо всех ресурсах добавляются в специальный файл R.jar, который затем используется при работе с ресурсами

Применение ресурсов

Существует два способа доступа к ресурсам: в файле исходного кода и в файле xml.

Ссылка на ресурсы в коде

Тип ресурса в данной записи ссылается на одно из пространств (вложенных классов), определенных в файле R.java, которые имеют соответствующие им типы в xml:

- R.drawable (ему соответствует тип в xml drawable)
- R.id (id)
- R.layout (layout)
- R.string (string)
- R.attr (attr)
- R.plural (plurals)
- R.array (string-array)

Например, для установки ресурса activity_main.xml в качестве графического интерфейса в коде MainActivity в методе onCreate() есть такая строка:

setContentView(R.layout.activity_main);

Через выражение R.layout.activity_main мы и ссылаемся на ресурс activity_main.xml, где layout - тип ресурса, а activity_main - имя ресурса.

Подобным образом мы можем получать другие ресурсы. Например, в файле res/values/strings.xml определен ресурс app_name:

```
<resources>
    <string name="app_name">ViewApp</string>
</resources>
```

Этот ресурс ссылается на строку. Чтобы получить ссылку на данный ресурс в коде java, мы можем использовать выражение R.string.app_name.

Доступ в файле xml

Нередко возникает необходимость ссылаться на ресурс в файле xml, например, в файле, который определяет визуальный интерфейс, к примеру, в activity_main.xml. Ссылки на ресурсы в файлах xml имеют следующую формализованную форму: @[имя_пакета:]тип_ресурса/имя_ресурса

- имя_пакета представляет имя пакета, в котором ресурс находится (указывать необязательно, если ресурс находится в том же пакете)
- тип_ресурса представляет подкласс, определенный в классе R для типа ресурса
- имя_ресурса имя файла ресурса без расширения или значение атрибута android:name в XML-элементе (для простых значений).

Например, мы хотим вывести в элемент TextView строку, которая определена в виде ресурса в файле strings.xml:

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/app_name" />
```

В данном случае свойство text в качестве значения будет получать значение строкового ресурса app_name.

Метод getResources

Для получения ресурсов в классе Activity мы можем использовать метод getResources(), который возвращает объект android.content.res.Resources. Но чтобы получить сам ресурс, нам надо у полученного объекта Resources вызвать один из методов. Некоторые из его методов:

- getString(): возвращает строку из файла strings.xml по числовому идентификатору
- getDimension(): возвращает числовое значение - ресурс dimen
- getDrawable(): возвращает графический файл в виде объекта Drawable
- getBoolean(): возвращает значение boolean
- getColor(): возвращает определение цвета
- getColorStateList(): возвращает объект ColorStateList - набор цветов
- getFont(): возвращает определение шрифта в виде объекта Typeface
- getFloat(): возвращает значение float
- getLayout(): возвращает объект XmlResourceParser, связанный с файлом layout

Это только некоторые методы. Но все они в качестве параметра принимают идентификатор ресурса, который надо получить. Вкратце рассмотрим их применение. Возьмем тот же файл res/values/strings.xml в качестве источника ресурсов, который в моем случае выглядит так:

```
<resources>
    <string name="app_name">ViewApp</string>
</resources>
```

И изменим код MainActivity:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);
        // получение ресурсов из файла values/strings.xml
        String app_name = getResources().getString(R.string.app_name);

        TextView textView = new TextView(this);
        textView.setTextSize(30);
        textView.setText(app_name);

        setContentView(textView);
```

```
    }
}
```

Здесь, используя метод getResources() получаем все ресурсы и затем используем их для установки значений свойств графических элементов. При запуске приложения мы увидим применение полученных ресурсов:

ViewApp

Использование ресурсов в Android Studio

Подобным образом мы можем программно получать и другие ресурсы и использоват их в приложении. Однако следует отметить, что в данном случае нам не нужно использовать метод getResources() и вообще производить какие-то определенные действия по получению ресурса, поскольку метод setText() класса TextView поддерживает прямую установку текста по идентификатору ресурса:

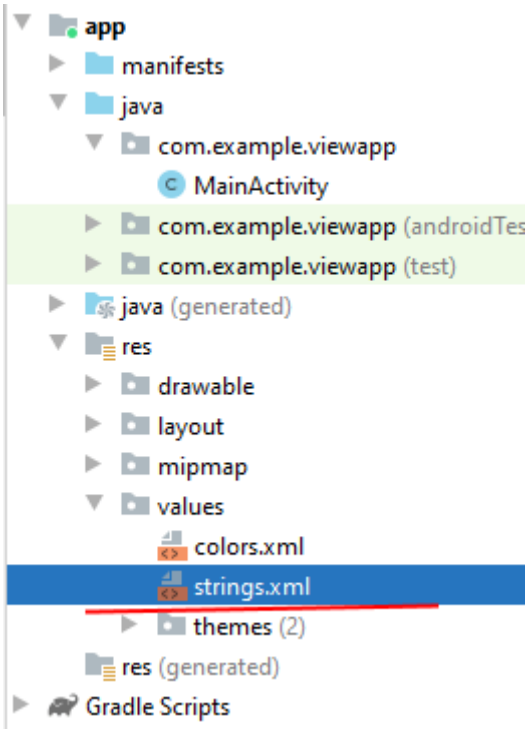
```
textView.setText(R.string.app_name);
```

Ресурсы строк

Ресурсы строк - один из важных компонентов приложения. Мы используем их при выведении названия приложения, различного текста, например, текста кнопок и т.д.

XML-файлы, представляющие собой ресурсы строк, находятся в проекте в папке res/values. По умолчанию ресурсы строк находятся в файле strings.xml, который может выглядеть следующим образом:

```
<resources>
    <string name="app_name">ViewApp</string>
</resources>
```



Ресурсы строк в Android и Java

В самом простом виде этот файл определяет один ресурс "app_name", который устанавливает название приложения, которое мы видим в заголовке приложения на экране устройства. Но естественно мы можем определить любые строковые ресурсы. Каждый отдельный ресурс определяется с помощью элемента string, а его атрибут name содержит название ресурса.

Затем в приложении в файлах кода мы можем ссылаться на эти ресурсы:

R.string.app_name

Например, в коде Java:

```
String application_name =  getResources().getString(R.string.app_name);
```

Либо в xml-файле:

```
@string/app_name
```

Например, изменим файл **res/values/strings.xml** следующим образом:

```
<resources>
    <string name="app_name">ViewApp</string>
    <string name="message">Hello Android!</string>
</resources>
```

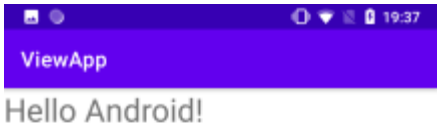
Здесь добавлен ресурс message со значением "Hello Android!".

Теперь используем ресурс в файле activity_main.xml:

```
<?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"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/message"
        android:textSize="30sp"
```

```
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

С помощью выражения @string/message передаем атрибуту android:text значение из ресурса.



Ресурсы строк из strings.xml в Android Studio и Java

Аналогично мы могли бы использовать ресурс в коде Activity:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
```

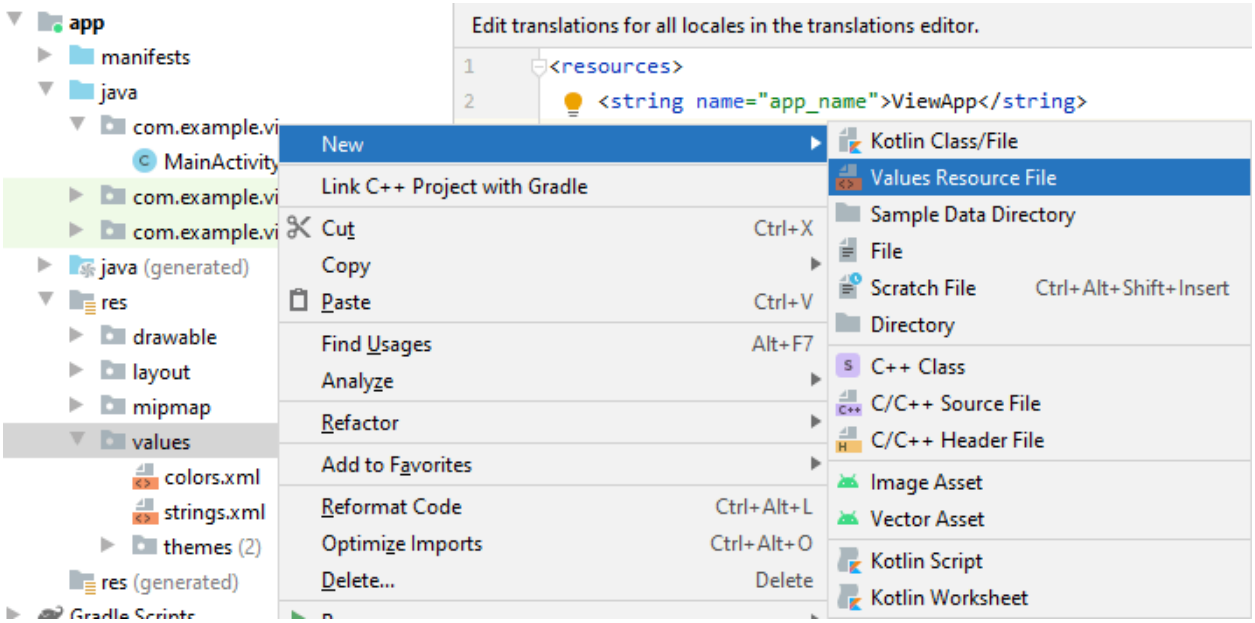
```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // получаем элемент textView
        TextView textView = findViewById(R.id.textView);
        // переустанавливаем у него текст
        textView.setText(R.string.message);
    }
}
```

Если нам в принципе надо получить ресурс в коде Java (необязательно для установки текста в TextView), то в этом случае можно использовать метод getResources().getString(идентификатор_ресурса)

```
String message = getResources().getString(R.string.message);
```

Хотя по умолчанию для ресурсов строк применяется файл strings.xml, но разработчики могут добавлять дополнительные файлы ресурсов в каталог проекта res/values. При этом достаточно соблюдать структуру файла: он должен иметь корневой узел <resources> и иметь один или несколько элементов <string>.

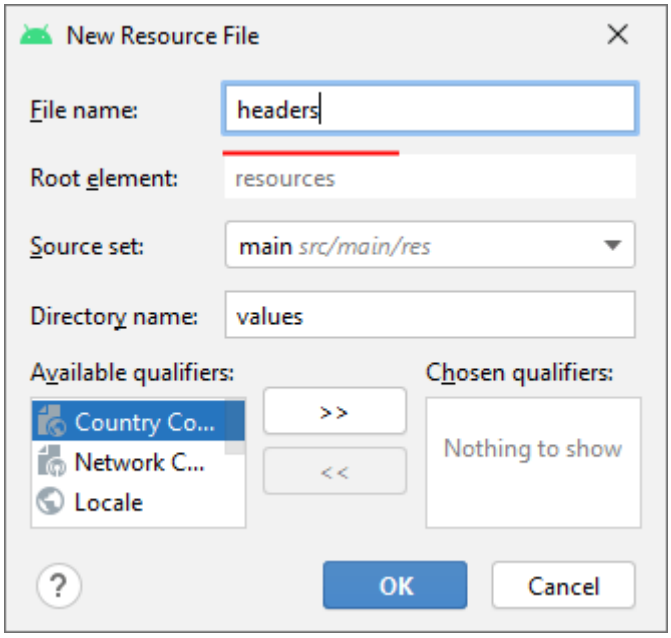
Так, нажмем на папку res/values правой кнопкой мыши и в появившемся списке выберем пункт **New -> Value Resource File**:



Добавление ресурса строк в Android Studio

Причем следует отметить, что данный тип файлов будет характерен для любого типа ресурсов, который добавляется в папку **res/values**.

После этого нам будет предложено определить для файла имя:



String resources in Android Studio

Назовем, к примеру, headers (название файла произвольное), а для всех остальных полей оставим значения по умолчанию. И в папку res/values будет добавлен новый файл headers.xml. Определим в нем пару ресурсов:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="welcome">Добро пожаловать</string>
    <string name="click_button">Нажмите на кнопку</string>
</resources>
```

И после этого мы сможем использовать определенные здесь ресурсы в коде Activity или в файле layout.

Форматирование строк

Android позволяет применять к ресурсам строк форматирование. Например, изменим файл strings.xml:

```
<resources>
    <string name="app_name">ViewApp</string>
    <string name="message">Hello Android!</string>
    <string name="welcome_message">Добро пожаловать %1$s! Уже %2$d : %3$d</string>
</resources>
```

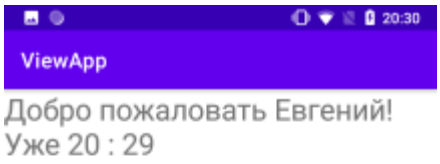
Третий ресурс - welcome_message представляет строку с форматированием. Так, она содержит такие символы как %1$s, %2$d и %3$d. Что они означают? %1$s указывает, что это первый аргумент, а символ "s" говорит, что этот аргумент представляет строку. %2$d представляет второй аргумент, а символ "d" в конце указывает, что это будет целое число. Аналогично %3$d указывает, что это третий аргумент, который представляет целое число.

Получим ресурс в коде Java

```
package com.example.viewapp;
```

```
import androidx.appcompat.app.AppCompatActivity;

import android.content.res.Resources;
import android.os.Bundle;
import android.widget.TextView;

import java.util.Calendar;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);
        Resources res = getResources();

        String userName = "Евгений";
        Calendar calendar = Calendar.getInstance();
        int hour = calendar.get(Calendar.HOUR_OF_DAY);
        int minute = calendar.get(Calendar.MINUTE);

        String text = getString(R.string.welcome_message, userName, hour, minute);
        TextView textView = new TextView(this);
        textView.setText(text);
        textView.setTextSize(28);
        setContentView(textView);
    }
}
```

Метод getString(R.string.welcome_message, userName, hour, minute) получает ресурс welcome_message и в качестве последующих параметров передает его аргументам значения. Для первого аргумента-строки используется переменная userName, а для второго и третьего аргументов передаем текущее количество часов и минут, полученных с помощью класса Calendar.

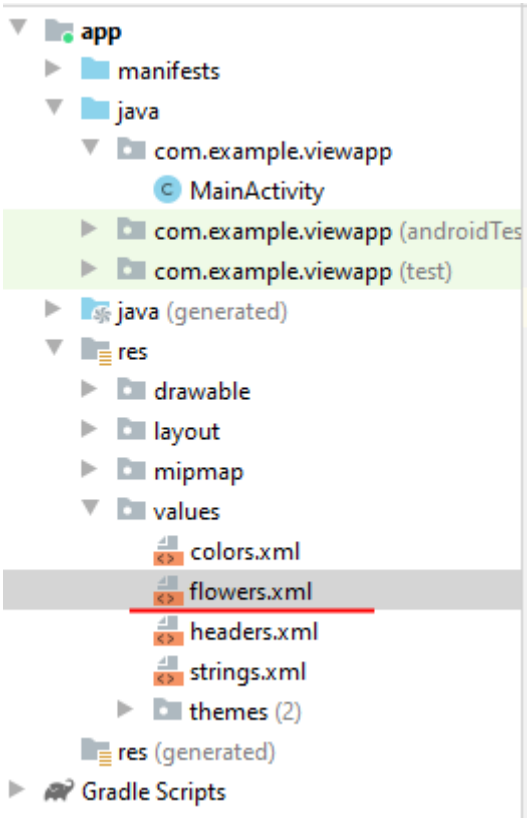


Форматирование ресурса в string.xml в Android и Java

Ресурсы Plurals

Plurals представляют еще один вид набора строк. Он предназначен для описания количества элементов. Для чего это надо? К примеру, возьмем существительное: нередко оно изменяет окончание в зависимости от числительного, которое с ним употребляется: 1 цветок, 2 цветка, 5 цветков. Для подобных случаев и используется ресурс plurals.

Посмотрим на примере. Добавим в папку res/values новый ресурс. Назовем его **flowers**:



Ресурс Plurals в Android Studio

Изменим его содержимое следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals name="flowers">
        <item quantity="one">%d цветок</item>
        <item quantity="few">%d цветка</item>
        <item quantity="many">%d цветков</item>
    </plurals>
</resources>
```

Для задания ресурса используется элемент <plurals>, для которого существует атрибут name, получающий в качестве значения произвольное название, по которому потом ссылаются на данный ресурс.

Сами наборы строк вводятся дочерними элементами <item>. Этот элемент имеет атрибут quantity, который имеет значение, указывающее, когда эта строка используется. Данный атрибут может принимать следующие значения:

- zero: строка для количества в размере 0
- one: строка для количества в размере 1 (для русского языка - для задания всех количеств, оканчивающихся на 1, кроме 11)
- two: строка для количества в размере 2
- few: строка для небольшого количества
- many: строка для больших количеств
- other: все остальные случаи

Причем в данном случае многое зависит от конкретного языка. А система сама позволяет определить, какое значение брать для того или иного числа.

Использование данного ресурса возможно толко в коде java. Поэтому изменим код MainActivity:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.TextView;
```

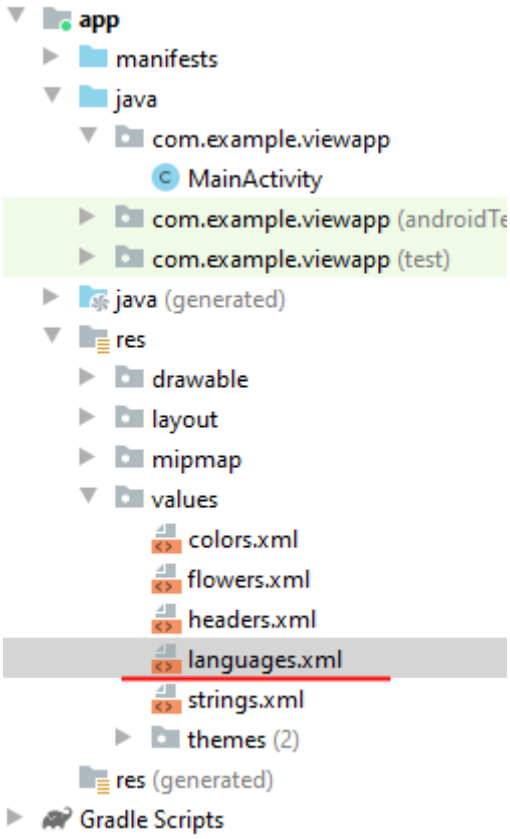
```
public class MainActivity extends AppCompatActivity {


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);
        String rose = getResources().getQuantityString(R.plurals.flowers, 21, 21);


        TextView textView = new TextView(this);
        textView.setText(rose);
        textView.setTextSize(26);
        setContentView(textView);
    }
}
```

С помощью метода **getQuantityString** мы получаем значение ресурса. Первым параметром передаем идентификатор ресурса. Вторым параметром идет значение. для которого нужно найти нужную строку. Третий параметр представляет собой значение, которое будет вставляться на место плейсхолдера %d. То есть мы получаем строку для числа 21.

Определение ресурса plurals в Android

string array

Еще одним видом строковых ресурсов является string-array или массив строк. Например, добавим в папку res/values новый файл, который назовем **languages.xml**:



Определение ресурса string array в Android и Java

Пусть он будет содержать следующий код:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="languages">
        <item>Java</item>
        <item>Kotlin</item>
        <item>Dart</item>
    </string-array>
</resources>
```

Ресурс задается с помощью элемента <string-array>. Фактически он определяет набор строк. А каждая отдельная строка задается с помощью элемента <item>

В файле MainActivity.java определим код для получения значений из этого ресурса:

```
package com.example.viewapp;


import androidx.appcompat.app.AppCompatActivity;


import android.content.res.Resources;
import android.os.Bundle;
import android.widget.TextView;


public class MainActivity extends AppCompatActivity {


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);
        Resources res = getResources();
        String[] languages = res.getStringArray(R.array.languages);
        String allLangs = "";
        for (String lang: languages) {
            allLangs += lang + " ";
        }
TextView textView = new TextView(this);
        textView.setText(allLangs);
        textView.setTextSize(28);
        setContentView(textView);
    }
}
```

С помощью метода getStringArray получаем ресурс в массив строк и затем с помощью цикла складываем из массива одну строку и передаем ее в TextView.

Получение ресурса string array и getStringArray в Android и Java

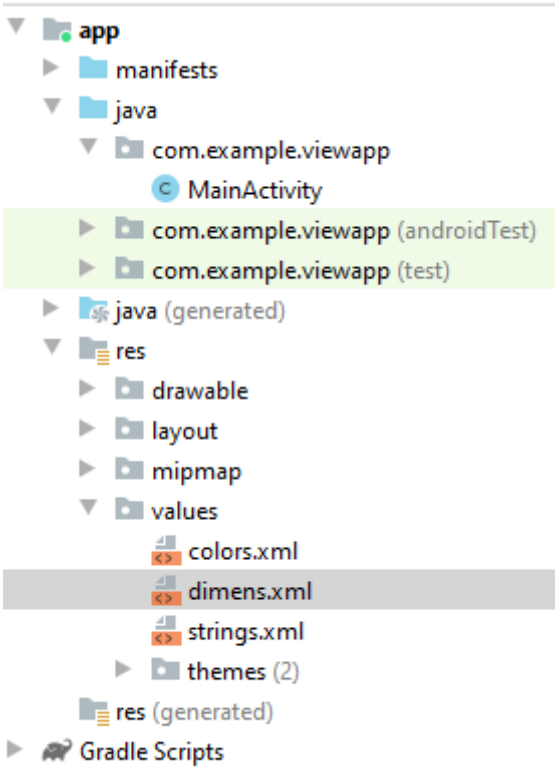
Ресурсы dimension

Определение размеров должно находиться в папке res/values в файле с любым произвольным именем. Общий синтаксис определения ресурса следующий:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="имя_ресурса">используемый_размер</dimen>
</resources>
```

Как и другие ресурсы, ресурс dimension определяется в корневом элементе <resources>. Тег <dimen> обозначает ресурс и в качестве значния принимает некоторое значение размера в одной из принятых единиц измерения (dp, sp, pt, px, mm, in). Более подробно установка размеров рассматривалась в одной из прошлых практических занятий.

Так, добавим в Android Studio в папку res/values новый элемент Values Resources File, который назовем dimens.xml.



Ресурс Dimension в Android Studio и Java, dimens.xml

Определим в нем следующее содержимое:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="horizontal_margin">64dp</dimen>
    <dimen name="vertical_margin">32dp</dimen>
    <dimen name="text_size">32sp</dimen>
</resources>
```

Здесь определены два ресурса для отступов horizontal_margin и vertical_margin, которые хранят соответственно значения 64dp и 32dp, и ресурс text_size, который хранит высоту шрифта - 32sp. Названия ресурсов могут быть произвольными.

Используем ресурс в файле activity_main.xml:

```
<?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"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello Android"
        android:background="#eaeaea"

        android:layout_marginTop="@dimen/vertical_margin"
        android:layout_marginLeft="@dimen/horizontal_margin"
        android:textSize="@dimen/text_size"

        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Ресурсы dimension используются для таких атриубтов визуальных элементов, которые в качестве значения требуют указание числового значения. Например, атриубут android:layout_height или android:textSize. Для получения ресурса в xml после "@dimen/" указывается имя ресурса.

Dimensions в Android и Java

Для получения ресурсов в коде java применяется метод getDimension() класса Resources. Например, определим в коде Java аналогичный визуальный интерфейс:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;

import android.content.res.Resources;
import android.os.Bundle;
import android.util.TypedValue;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);
        // получаем ресурсы
        Resources resources = getResources();
        float textSize = resources.getDimension(R.dimen.text_size);
        int hMargin = (int)resources.getDimension(R.dimen.horizontal_margin);
        int vMargin = (int)resources.getDimension(R.dimen.vertical_margin);

        ConstraintLayout constraintLayout = new ConstraintLayout(this);

        ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams
                (ConstraintLayout.LayoutParams.WRAP_CONTENT ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
        layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
        layoutParams.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;

        TextView textView = new TextView(this);
        textView.setText("Hello Android");
        textView.setBackgroundColor(0XFFEAEAEA);
        // устанавливаем размер шрифт по ресурсу
        textView.setTextSize(TypedValue.COMPLEX_UNIT_PX, textSize);
        // устанавливаем отступы пв соответствии с ресурсами
        layoutParams.setMargins(hMargin, vMargin, hMargin, vMargin);

        textView.setLayoutParams(layoutParams);
        constraintLayout.addView(textView);
```

```
        setContentView(constraintLayout);
    }
}
```

При программном получении ресурсов dimen с помощью метода getDimension() следует учитывать, что он возвращает значения в физических пикселях, даже если в файле ресурсов мы определили значения в других величинах (32sp, 64dp). В большинстве случае это не вызовет каких-то трудностей, поскольку большинство методов в Java принимают именно пиксели, а не dp или другие единицы, как например, метод setMargins(), который устанавливает отступы.

Однако метод setTextSize() принимает именно sp, поэтому с помощью дополнительного параметра необходимо указать, что в данном случае имеются в виду физические пиксели, а не sp:

```
textView.setTextSize(TypedValue.COMPLEX_UNIT_PX, textSize);
```

Либо, как вариант, с помощью класса TypedValue программно перевести физические пиксели в sp или другую единицу измерения.

Ресурсы Color и установка цвета

В приложении Android также можно определять ресурсы цветов (Color). Они должны храниться в файле по пути res/values и также, как и ресурсы строк, заключены в тег <resources>. Так, по умолчанию при создании самого простого проекта в папку res/values добавляется файл colors.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
```

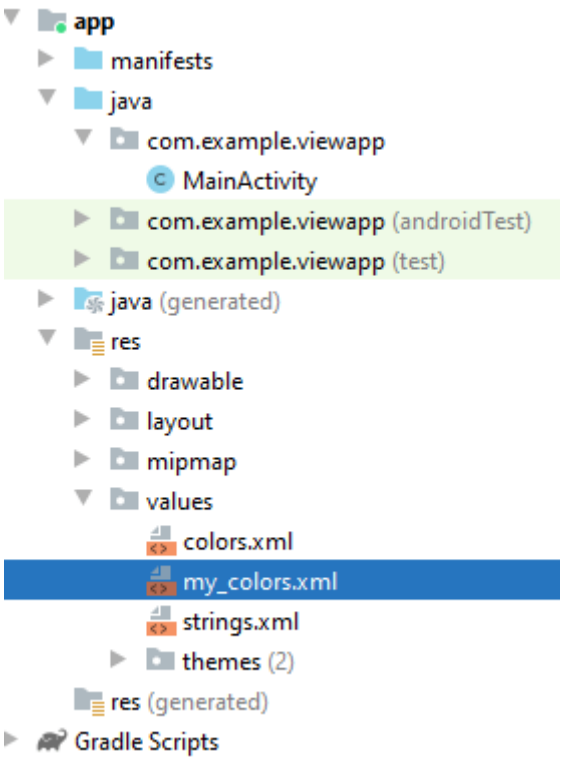
```
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
</resources>
```

Цвет определяется с помощью элемента <color>. Его атрибут name устанавливает название цвета, которое будет использоваться в приложении, а шестнадцатеричное число - значение цвета.

Для задания цветовых ресурсов можно использовать следующие форматы:

- #RGB (#F00 - 12-битное значение)
- #ARGB (#8F00 - 12-битное значение с добавлением альфа-канала)
- #RRGGBB (#FF00FF - 24-битное значение)
- #AARRGGBB (#80FF00FF - 24-битное значение с добавлением альфа-канала)

Чтобы не трогать и не портить данный файл, определим свой новый файл ресурсов и для этого добавим в папку res/values новый файл ресурсов, который назовем **my_colors.xml**.



Использование цвета в Android Studio и ресурсы color

Изменим файл my_colors.xml, добавив в него пару цветов:

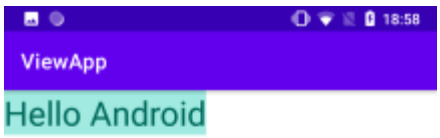
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="textViewBackColor">#A0EAE1</color>
    <color name="textViewFontColor">#00695C</color>
</resources>
```

Применим цвета в файле activity_main.xml:

```
<?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"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
```

```
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello Android"

        android:background="@color/textViewBackColor"
        android:textColor="@color/textViewFontColor"

        android:textSize="32sp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

С помощью атрибута android:textColor устанавливается цвет текста в TextView, а атрибут android:background устанавливает фон TextView. В качестве значения они используют цвет, например, в том же шестнадцатеричном формате. Для получения самого цвета после "@color/" указывается имя ресурса.



Использование цвета в Android и Java и ресурсы color

Также можно использовать цветовые ресурсы в коде MainActivity:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;

import android.content.res.Resources;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        //setContentView(R.layout.activity_main);
        // получаем ресурсы
        Resources resources = getResources();
        int textColor = resources.getColor(R.color.textViewFontColor,  null);
        int backgroundColor = resources.getColor(R.color.textViewBackColor,  null);

        ConstraintLayout constraintLayout = new ConstraintLayout(this);

        ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams
                (ConstraintLayout.LayoutParams.WRAP_CONTENT ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
        layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
        layoutParams.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;

        TextView textView = new TextView(this);
        textView.setText("Hello Android");
        textView.setTextSize(32);

        // используем ресурсы color
        textView.setTextColor(textColor);
        textView.setBackgroundColor(backgroundColor);

        textView.setLayoutParams(layoutParams);
        constraintLayout.addView(textView);

        setContentView(constraintLayout);
    }
}
```

Для получения цвета из ресурсов применяется метод **resources.getColor(),** который принимает два параметра. Первый параметр - идентификатор ресурса, цвет которого надо получить. Второй параметр представляет тему. Но поскольку в данном случае тема не важна, для этого параметра передаем значение null

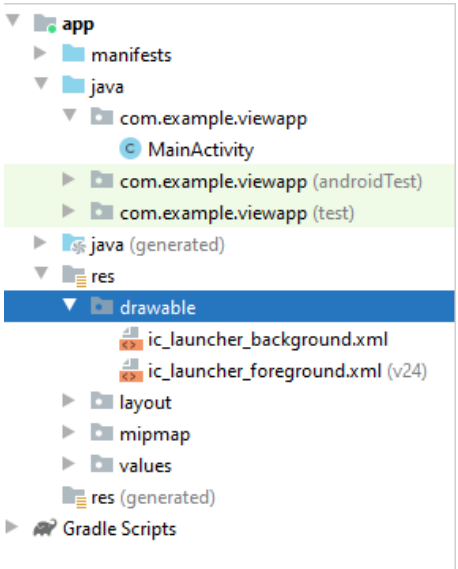
Следует учитывать, что метод **resources.getColor()** с двумя параметрами, который использован выше, доступен, если для минимальная версия Android не ниже Android 6.0 (или Android 23). Однако минимальная версия Android ниже, то можно использовать устаревшую версию с одним параметром:

```
int textColor = resources.getColor(R.color.textViewFontColor);

// вместо

//int textColor = resources.getColor(R.color.textViewFontColor,  null);
```

Работа с изображениями

Ресурсы изображений

Одним из наиболее распространенных источников ресурсов являются файлы изображений. Android поддерживает следующие форматы файлов: .png (предпочтителен), .jpg (приемлем), .gif (нежелателен). Для графических файлов в проекте уже по умолчанию создана папка **res/drawable**. По умолчанию она уже содержит ряд файлов - пару файлов иконок:



Работа с изображениями в Android

При добавлении графических файлов в эту папку для каждого из них Android создает ресурс **Drawable**. После этого мы можем обратиться к ресурсу следующим образом в коде Java:

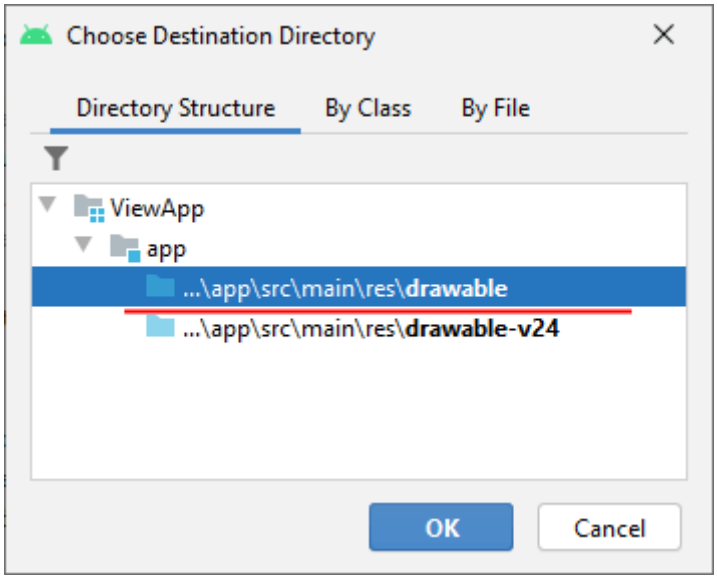
R.drawable.имя_файла

Или в коде xml:

@[имя_пакета:]drawable/имя_файла

Например, добавим в проект в папку **res/drawable** какой-нибудь файл изображения. Для этого скопируем на жестком диске какой-нибудь файл с расширением png или jpg и вставим его в папку **res/drawable** (для копирования в проект используется простой Copy-Paste)

Далее нам будет предложено выбрать папку - **drawable** или **drawable-24**. Для добавления обычных файлов изображений выберем **drawable**:

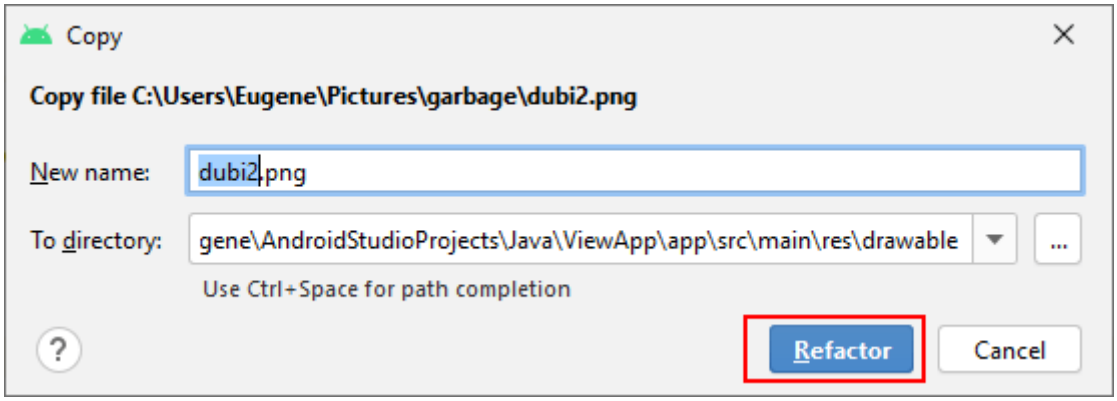


Добавление изображений в папку drawable Android Studio

Здесь сразу стоит учесть, что файл изображения будет добавляться в приложение, тем самым увеличивая его размер. Кроме того, большие изображения отрицательно влияют на производительность. Поэтому лучше использовать небольшие и оптимизированные (сжатые) графические файлы. Хотя, также стоит отметить, что все файлы изображений, которые

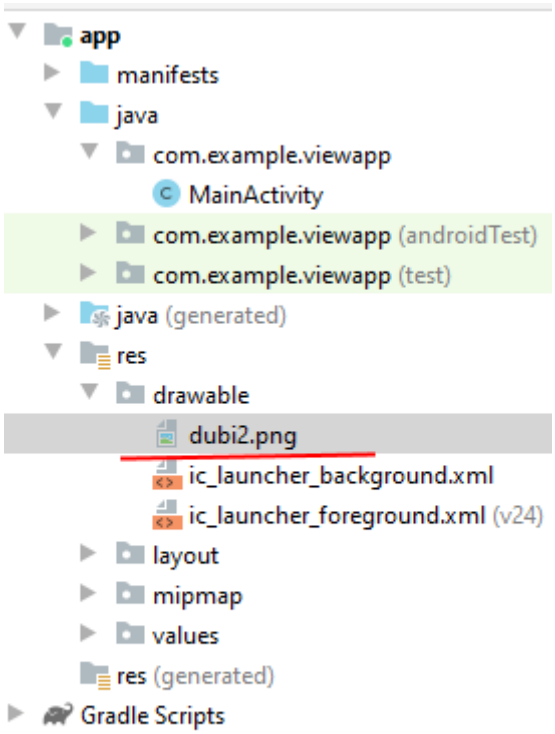
добавляются в эту папку, могут автоматически оптимизироваться с помощью утилиты **aapt** во время построения проекта. Это позволяет уменьшить размер файла без потери качества.

При копировании файла нам будет предложено установить для него новое имя.



Ресурсы drawable в Android и Java

Можно изменить название файла, а можно оставить так как есть. В моем случае файл называется **dubi2.png**. И затем нажмем на кнопку Refactor. И после этого в папку drawable будет добавлен выбранный нами файл изображения.



Добавление изображений в Android Studio

Для работы с изображениями в Android можно использовать различные элементы, но непосредственно для вывода изображений предназначен ImageView. Поэтому изменим файл activity_main.xml следующим образом:

```
<?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"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/dubi2"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

В данном случае для отображения файла в ImageView у элемента устанавливается атрибут **android:src**. В его значении указывается имя графического ресурса, которое совпадает с именем файла без расширения. И после этого уже в Preview или в режиме дизайнере в Android Studio можно будет увидеть применение изображения, либо при запуске приложения:

ImageView и drawable в Android Studio и Java

Если бы мы создавали ImageView в коде java и из кода применяли бы ресурс, то activity могла бы выглядеть так:

```
package com.example.viewapp;


import androidx.appcompat.app.AppCompatActivity;

import androidx.constraintlayout.widget.ConstraintLayout;

import android.os.Bundle;

import android.widget.ImageView;


public class MainActivity extends AppCompatActivity {


    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);

        ConstraintLayout constraintLayout = new ConstraintLayout(this);
        ImageView imageView = new ImageView(this);
        // применяем ресурс
        imageView.setImageResource(R.drawable.dubi2);

        ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams
                (ConstraintLayout.LayoutParams.WRAP_CONTENT ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
        layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
        layoutParams.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
        imageView.setLayoutParams(layoutParams);
        constraintLayout.addView(imageView);

        setContentView(constraintLayout);
    }
}
```

В данном случае ресурс drawable напрямую передается в метод imageView.setImageResource(), и таким образом устанавливается изображение. В результате мы получим тот же результат.

```
imageView.setImageResource(R.drawable.dubi2);
```

Однако может возникнуть необходимость как-то обработать ресурс перед использованием или использовать его в других сценариях. В этом случае мы можем сначала получить его как объект Drawable и затем использовать для наших задач:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;
import androidx.core.content.res.ResourcesCompat;

import android.content.res.Resources;
import android.graphics.drawable.Drawable;
import android.os.Bundle;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);
        ConstraintLayout constraintLayout = new ConstraintLayout(this);
        ImageView imageView = new ImageView(this);
        Resources res = getResources();
        Drawable drawable = ResourcesCompat.getDrawable(res, R.drawable.dubi2,
null);
        // применяем ресурс
        imageView.setImageDrawable(drawable);

        ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams
                (ConstraintLayout.LayoutParams.WRAP_CONTENT ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
        layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
```

```
        layoutParams.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
        imageView.setLayoutParams(layoutParams);
        constraintLayout.addView(imageView);


        setContentView(constraintLayout);
    }
}
```

Для получения ресурса применяется метод ResourcesCompat.getDrawable(), в который передается объект Resources, идентификатор ресурса и тема. В данном случае тема нам не важна, поэтому для нее передаем значение null. Возвращается ресурс в виде объекта **Drawable:**

```
Drawable drawable = ResourcesCompat.getDrawable(res, R.drawable.dubi2, null);
```

Затем, например, можно также передать ресурс объекту ImageView через его метод setImageDrawable()

```
imageView.setImageDrawable(drawable);
```

ImageView

В прошлом материале было рассмотрено, как выводить изображения с помощью элемента ImageView. Теперь рассмотрим некоторые дополнительные моменты по работе с этим элементом.

Некоторые основные атрибуты элемента ImageView:

- **android:cropToPadding**: при значении true изображение обрезается в соответствии с установленными отступами

- **android:scaleType**: устанавливает, как изображение будет масштабироваться относительно границ элемента ImageView

Чтобы задать параметры масштабирования, используется одно из значений перечисления :

- ✓ CENTER: изображение центрируется по центру без масштабирования
- ✓ CENTER_CROP: изображение центрируется по центру и масштабируется с сохранением аспектного отношения между шириной и высотой. Если какая-то часть не помещается в пределы экрана, то она обрезается
- ✓ CENTER_INSIDE: изображение центрируется по центру и масштабируется с сохранением аспектного отношения между шириной и высотой, но ширина и высота не могут быть больше ширины и высоты ImageView
- ✓ FIT_CENTER: изображение масштабируется и центрируется
- ✓ FIT_START: изображение масштабируется и устанавливается в начало элемента (вверх при портретной ориентации и влево - при альбомной)
- ✓ FIT_END: изображение масштабируется и устанавливается в конец элемента (вниз при портретной ориентации и вправо - при альбомной)
- ✓ FIT_XY: изображение масштабируется без сохранения аспектного отношения межуд шириной и высотой, заполняя все пространство ImageView

- ✓ MATRIX: изображение масштабируется с применением матрицы изображения

- **android:src**: ресурс изображения

- **android:alpha**: устанавливает прозрачность (значение от 0.0 - полностью прозрачное до 1.0 - полностью видимо)

- **android:tint**: цвет, который используется для наложения на изображение

- android:tintMode: режим, который применяется для наложения цвета на изображения

Некоторые основные методы класса **ImageView**:

- **Drawable getDrawable()**: возвращает ресурс Drawable, который связан с данным ImageView (или null, если ресурс для ImageView не устанавлен)

- **ImageView.ScaleType getScaleType():** возвращает значение перечисления ImageView.ScaleType, которое указывает, как масштабируется изображение относительно границ элемента ImageView

- **void setImageDrawable(Drawable drawable):** устанавливает ресурс изображения с помощью объекта Drawable

- **void setImageResource(int resId):** устанавливает ресурс изображения с помощью идентификатора ресурса Drawable

- **void setImageURI(Uri uri):** устанавливает ресурс изображения с помощью адреса Uri этого ресурса

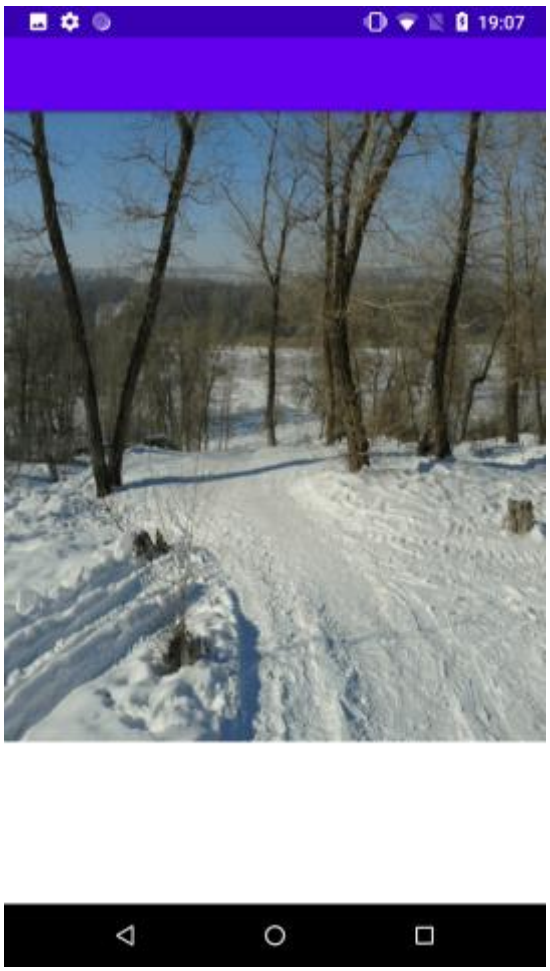
- **void setScaleType(ImageView.ScaleType scaleType):** задает масштабирование изображения

- **void setImageAlpha(int alpha):** задает прозрачность изображения - значение от 0.0 до 1.0

Например, установка значения FIT_XY для атрибута android:scaleType в файле **activity_main.xml**:

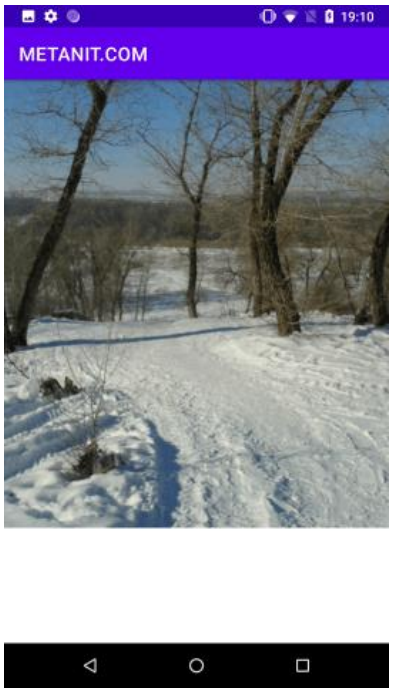
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/dubi2"
        android:scaleType="fitXY"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

В итоге изображение растянется по вертикали и горизонтали:



Элемент ImageView с масштабированием в Android

Для сравнения аналогичный пример с android:scaleType="center":



Элемент ImageView с центрированием в Android

Аналогичный пример в коде java:

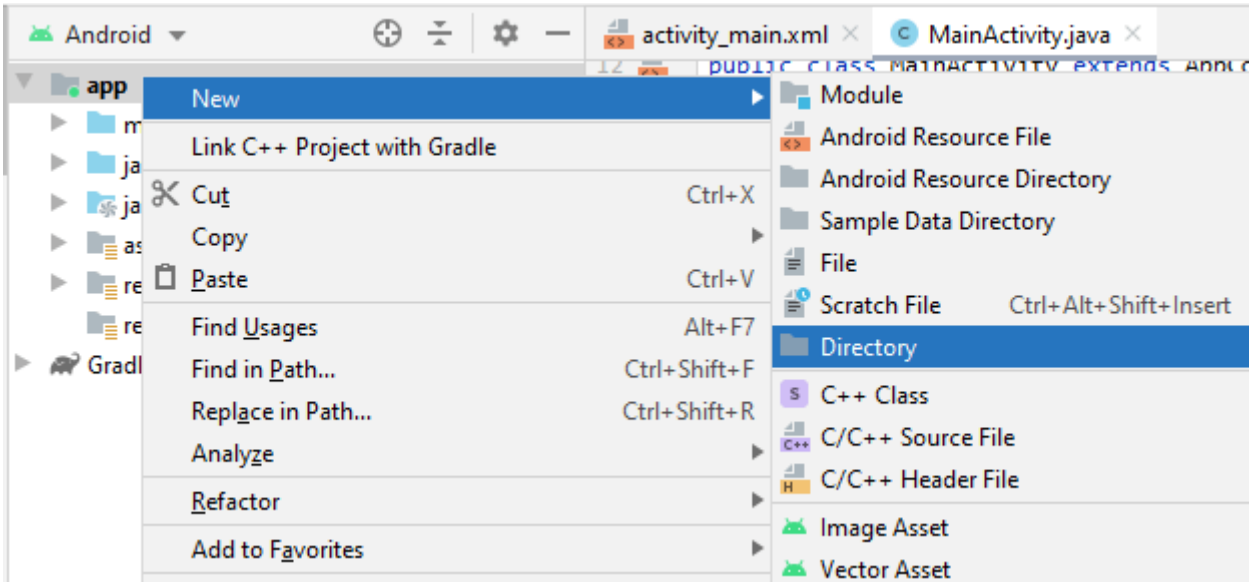
```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;
import android.os.Bundle;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);

        ConstraintLayout constraintLayout = new ConstraintLayout(this);
        ImageView imageView = new ImageView(this);
        imageView.setImageResource(R.drawable.dubi2);
        // задаем масштабирование
        imageView.setScaleType(ImageView.ScaleType.FIT_XY);

        ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams
                (ConstraintLayout.LayoutParams.WRAP_CONTENT ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
        layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
        layoutParams.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
        imageView.setLayoutParams(layoutParams);
        constraintLayout.addView(imageView);
```

```
        setContentView(constraintLayout);
    }
}
```

Изображения из папки assets

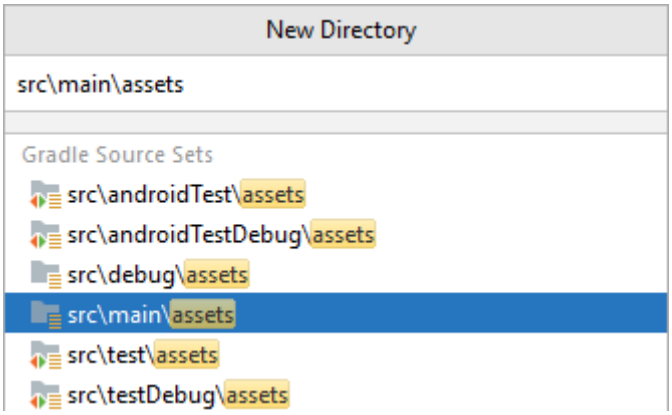
В прошлых материалах изображения в проекте помещались в папку res/drawables в качестве ресурсов и выводились в элемент ImageView. Однако изображения необязательно в принципе помещать именно в эту папку. Файлы также могут располагаться в папке assets. Рассмотрим, как работать с такими файлами изображений.

Вначале добавим в проект папку assets. Для этого в Android Studio нажмем на каталог app и в появившемся контекстном меню выберем **New -> Directory**:



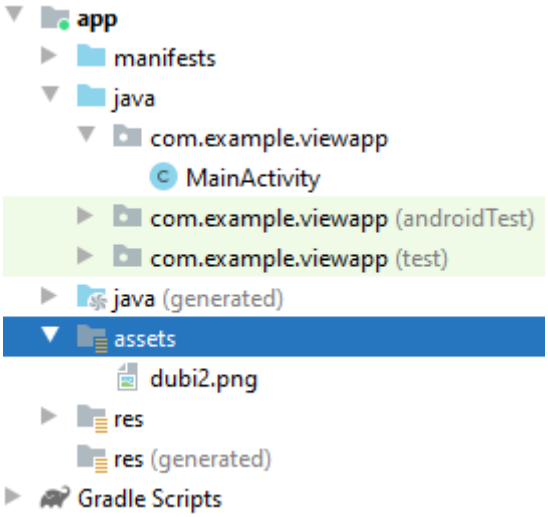
Добавление папки в Android Studio

Затем в появившемся окошке выберем пункт **src\main\assets** и нажмем на Enter для ее добавления в проект:



Добавление папки assets в Android Studio

Добавим в эту папку какое-нибудь изображение:



Drawables в assets в Android Studio и Java

Пусть в файле **activity_main.xml** будет определен элемент ImageView:

```
<?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"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ImageView
```

```
        android:id="@+id/image"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_margin="16dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Загрузим изображение из папки assets в элемент **ImageView** в **MainActivity**:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.graphics.drawable.Drawable;
import android.os.Bundle;
import android.widget.ImageView;

import java.io.IOException;
import java.io.InputStream;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
        ImageView imageView = findViewById(R.id.image) ;
        String filename = "dubi2.png";
        try(InputStream inputStream = getApplicationContext().getAssets().open(filename)){
            Drawable drawable = Drawable.createFromStream(inputStream, null);
            imageView.setImageDrawable(drawable);
            imageView.setScaleType(ImageView.ScaleType.FIT_XY);
        }
        catch (IOException e){
            e.printStackTrace();
        }
    }
}
```

Для загрузки файла необходиом получить поток InputStream с помощью выражения **getApplicationContext().getAssets().open(filename)**.

Вызов **Drawable.createFromStream(inputStream, null)** формирует объект Drawable из входного потока.

Метод **imageView.setImageDrawable(d)** загружает Drawable в **ImageView**.

Загрузка изображения из assets в Android

Задание

Создать примеры, реализующие функции ресурсов, изучить виды и работу с ресурсами. Реализовать два способа доступа к ресурсам: в файле исходного кода и в файле xml:

1. Реализовать пример использования ресурсов строк
2. Реализовать пример использования форматирование строк.
3. Реализовать пример использования ресурса Plurals
4. Реализовать пример использования ресурса string array
5. Реализовать пример использования ресурса dimension
6. Реализовать пример использования ресурса Color
7. Реализовать пример использования элемента ImageView и папки res/drawables
8. Реализовать пример загрузки изображения из папки assets в Android