Практическая работа 12

Методические материалы

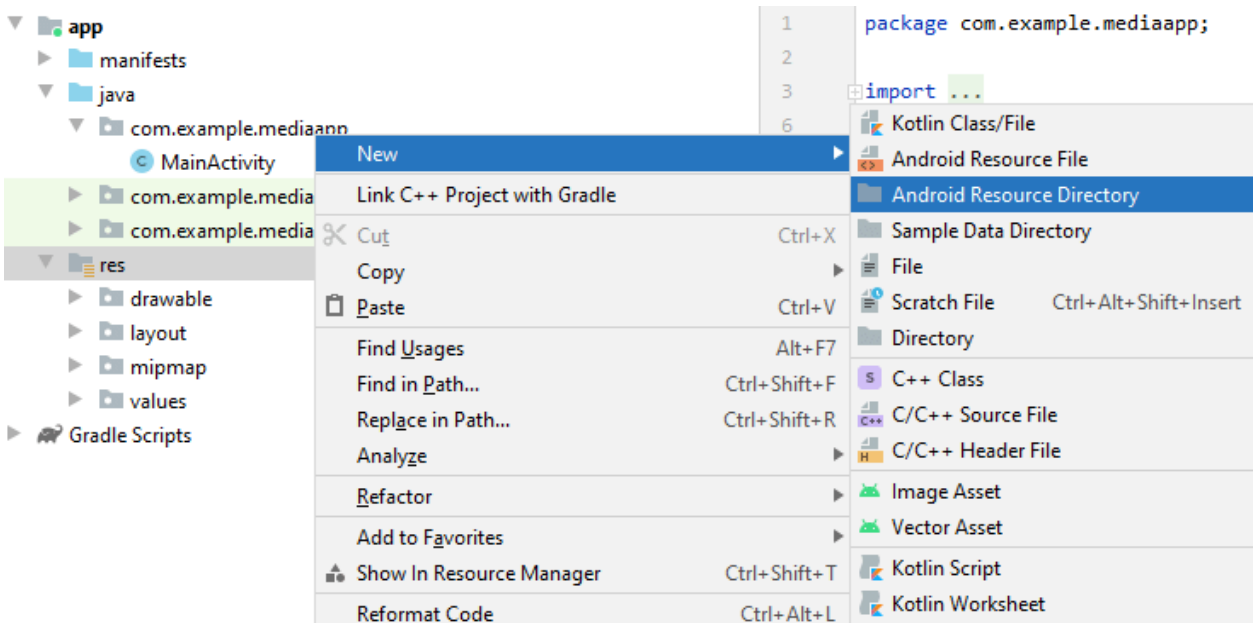
Работа с мультимедиа. Анимация. Сервисы. Диалоговые окна.

Работа с видео

Для работы с видеоматериалами в стандартном наборе виджетов Android определен класс VideoView, который позволяет воспроизводить видео.

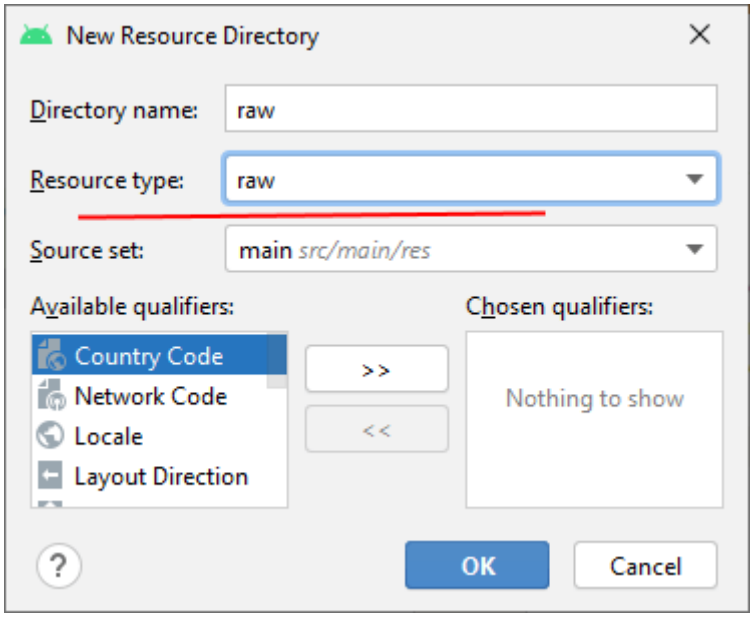
Какие типы видеофайлов можно использовать? Android поддерживает большинство распространенных типов видеофайлов, в частности, 3GPP (.3gp), WebM (.webm), Matroska (.mkv), MPEG-4 (.mp4).

VideoView может работать как с роликами, размещенными на мобильном устройстве, так и с видеоматериалами из сети. В данном случае используем видеоролик, размещенный локально. Для этого добавим в проект какой-нибудь видеоролик. Обычно видеоматериалы помещают в проекте в папку res/raw. По умолчанию проект не содержит подобной папки, поэтому добавим в каталог res подпапку raw. Для этого нажмем на папку res правой кнопкой мыши и в появившемся меню выберем **New -> Android Resource Directory:**



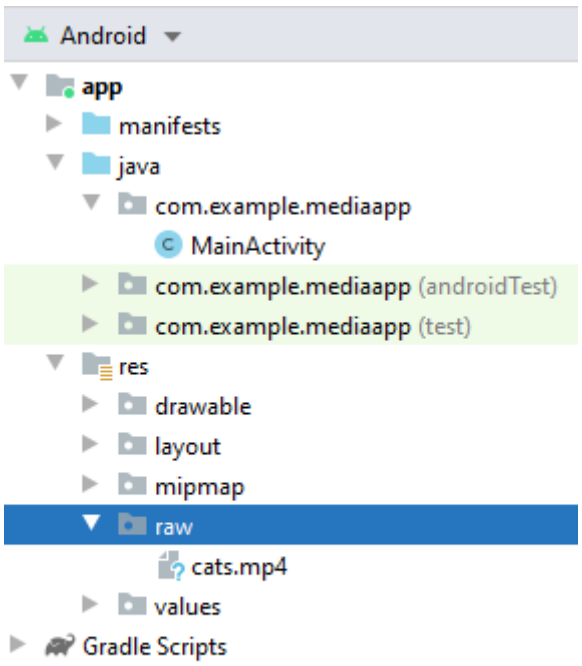
Добавление папки raw в проект Android Studio

Затем в появившемся окне в качестве типа папки укажем raw (что также будет использоваться в качестве названия папки):



Создание папки raw в проект Android Studio

После добавления папки raw скопируем в нее какой-нибудь видеофайл:



Добавление видеофайла в Android Studio

Теперь определим функционал для его воспроизведения. Для этого в файле **activity_main.xml** укажем следующий код:

```
<?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">

    <Button
        android:id="@+id/playButton"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Play"
        android:onClick="play"
```

```
    app:layout_constraintBottom_toTopOf="@id/videoPlayer"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toLeftOf="@id/pauseButton"
    app:layout_constraintTop_toTopOf="parent" />
<Button
    android:id="@+id/pauseButton"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Pause"
    android:onClick="pause"
    app:layout_constraintBottom_toTopOf="@id/videoPlayer"
    app:layout_constraintLeft_toRightOf="@id/playButton"
    app:layout_constraintRight_toLeftOf="@id/stopButton"
    app:layout_constraintTop_toTopOf="parent"/>
<Button
    android:id="@+id/stopButton"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Stop"
    android:onClick="stop"
    app:layout_constraintBottom_toTopOf="@id/videoPlayer"
    app:layout_constraintLeft_toRightOf="@id/pauseButton"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
<VideoView android:id="@+id/videoPlayer"
    android:layout_height="0dp"
    android:layout_width="0dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
```

```
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/playButton"/>

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

Для управления воспроизведением видео здесь определены три кнопки: для запуска видео, для паузы и для его остановки.

И также изменим код **MainActivity:**

```
package com.example.mediaapp;

import androidx.appcompat.app.AppCompatActivity;

import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.VideoView;

public class MainActivity extends AppCompatActivity {

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

        videoPlayer = findViewById(R.id.videoPlayer);
        Uri myVideoUri= Uri.parse( "android.resource://" + getPackageName() + "/"
+ R.raw.cats);
```

```
        videoPlayer.setVideoURI(myVideoUri);
    }

    public void play(View view){
        videoPlayer.start();
    }
    public void pause(View view){
        videoPlayer.pause();
    }
    public void stop(View view){
        videoPlayer.stopPlayback();
        videoPlayer.resume();
    }
}
```

Во-первых, чтобы управлять потоком воспроизведения, нам надо получить объект VideoView: videoPlayer = findViewById(R.id.videoPlayer);

Чтобы указать источник воспроизведения, необходим объект Uri. В данном случае с помощью выражения Uri myVideoUri= Uri.parse("android.resource://" + getPackageName() + "/" + R.raw.cats); получаем адрес видеоролика внутри пакета приложения.

Строка URI имеет ряд частей: сначала идет Uri-схема (http:// или как здесь android.resource://), затем название пакета, получаемое через метод getPackageName(), и далее непосредственно название ресурса видео из папки res/raw, которое совпадает с названием файла.

Затем этот Uri устанавливается у videoPlayera: videoPlayer.setVideoURI(myVideoUri);

Чтобы управлять видеопотоком, обработчики нажатия кнопок вызывают соответствующее действие:

```
public void play(View view){
    videoPlayer.start();
}
public void pause(View view){
    videoPlayer.pause();
}
public void stop(View view){
    videoPlayer.stopPlayback();
    videoPlayer.resume();
}
```

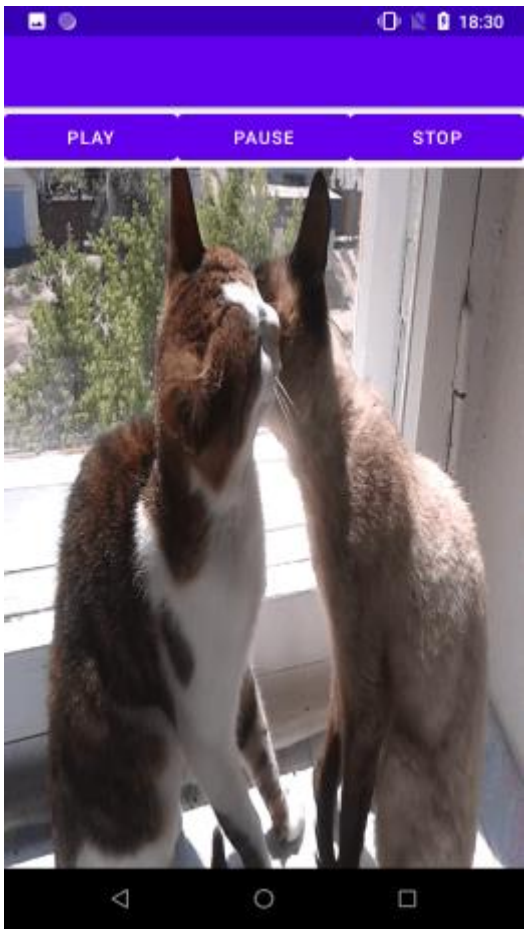
Метод **videoPlayer.start()** начинает или продолжает воспроизведение.

Метод **videoPlayer.pause()** приостанавливает видео.

Метод **videoPlayer.stopPlayback()** полностью останавливает видео.

Метод **videoPlayer.resume()** позволяет снова начать воспроизведение видео с начала после его полной остановки.

При запуске приложения мы сможем с помощью кнопок управлять воспроизведением:



Воспроизведение видео в Android и VideoView

MediaController

С помощью класса MediaController мы можем добавить к VideoView дополнительно элементы управления. Для этого изменим код MainActivity:

```
package com.example.mediaapp;

import androidx.appcompat.app.AppCompatActivity;

import android.net.Uri;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.VideoView;
import android.widget.MediaController;

public class MainActivity extends AppCompatActivity {

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


        videoPlayer = findViewById(R.id.videoPlayer);
        Uri myVideoUri= Uri.parse( "android.resource://" + getPackageName() + "/"
+ R.raw.cats);
        videoPlayer.setVideoURI(myVideoUri);
        MediaController mediaController = new MediaController(this);
        videoPlayer.setMediaController(mediaController);
        mediaController.setMediaPlayer(videoPlayer);
    }


    public void play(View view){
        videoPlayer.start();
    }
    public void pause(View view){
        videoPlayer.pause();
    }
    public void stop(View view){
        videoPlayer.stopPlayback();
        videoPlayer.resume();
```

```
    }
}
```

И если мы запустим приложения, то при касании по VideoView внизу появятся инструменты для управления видео. В прицнипе теперь и кнопки, которые мы создали ранее, не нужны:

VideoView и MediaController в Android

Воспроизведение файла из интернета

VideoView поддерживает воспроизведение файла из интернета. Но чтобы это стало возможно, необходимо в файле **AndroidManifest.xml** установить разрешение **android.permission.INTERNET**, так как мы получаем данные из интернета:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Далее изменим класс MainActivity:

```
package com.example.mediaapp;
```

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

import android.os.Bundle;
import android.view.View;
import android.widget.VideoView;

public class MainActivity extends AppCompatActivity {

    VideoView videoPlayer;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        videoPlayer = findViewById(R.id.videoPlayer);
        videoPlayer.setVideoPath("http://techslides.com/demos/sample-
videos/small.mp4");
    }

    public void play(View view){
        videoPlayer.start();
    }
    public void pause(View view){
        videoPlayer.pause();
    }
    public void stop(View view){
        videoPlayer.stopPlayback();
        videoPlayer.resume();
    }
}
```

Здесь нам надо в метод videoPlayer.setVideoPath() передать интернет-адрес воспроизводимого файла.

Воспроизведение аудио

Для воспроизведения музыки и других аудиоматериалов Android предоставляет класс **MediaPlayer**.

Чтобы воспроизводить аудио, MediaPlayer должен знать, какой именно ресурс (файл) нужно производить. Установить нужный ресурс для воспроизведения можно тремя способами:

- в метод **create()** объекта MediaPlayer передается id ресурса, представляющего аудиофайл
- в метод **create()** объекта MediaPlayer передается объект Uri, представляющего аудиофайл
- в метод **setDataSource()** объекта MediaPlayer передается полный путь к аудиофайлу

После установки ресурса вызывается метод **prepare()** или **prepareAsync()** (асинхронный вариант prepare()). Этот метод подготавливает аудиофайл к воспроизведению, извлекая из него первые секунды. Если мы воспроизводим файл из сети, то лучше использовать **prepareAsync().**

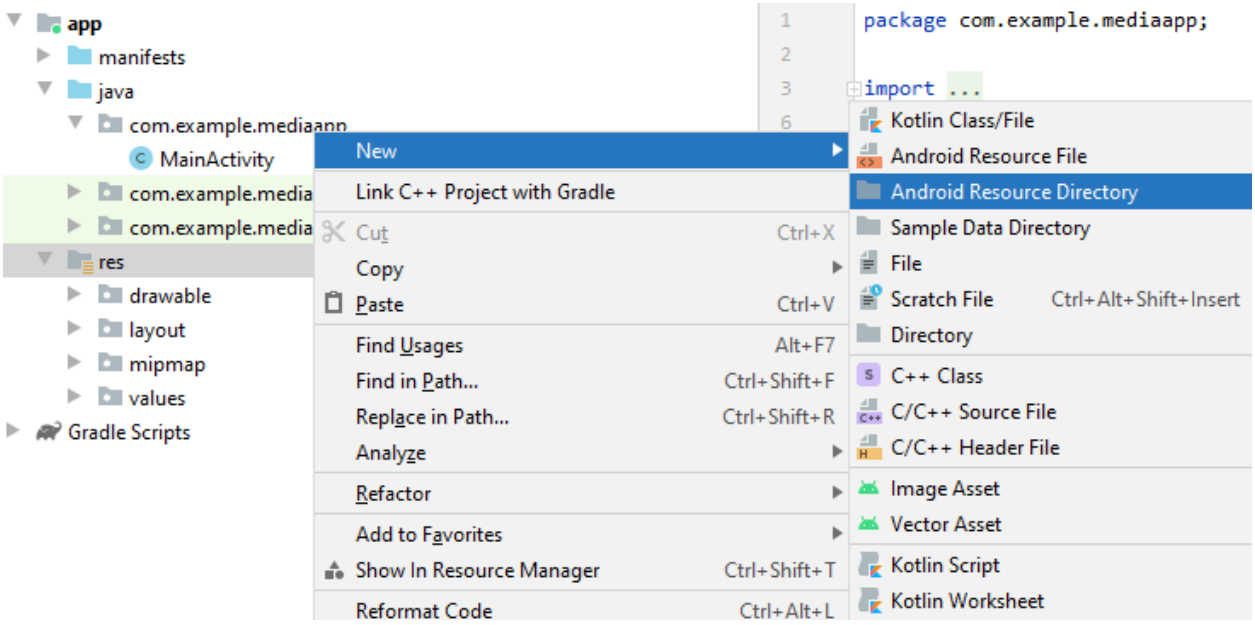
Для управления воспроизведением в классе MediaPlayer определены следующие методы:

- **start()**: запускает аудио

- **pause()**: приостанавливает воспроизведение

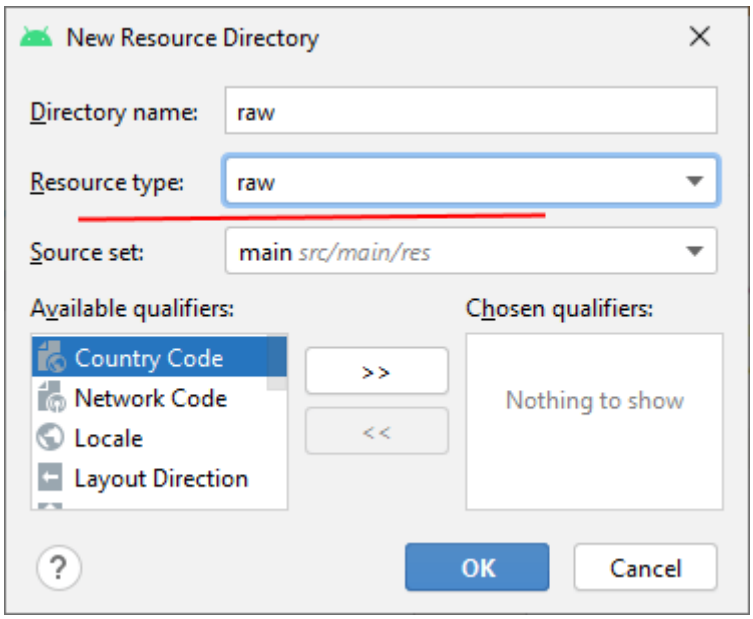
- **stop()**: полностью останавливает воспроизведение

Итак, создадим новый проект. Как и в случае с видео, аудиофайл должен находиться в папке **res/raw**, поэтому добавим в проект в Android Studio такую папку. Для этого нажмем на папку res правой кнопкой мыши и в появившемся меню выберем **New -> Android Resource Directory**:



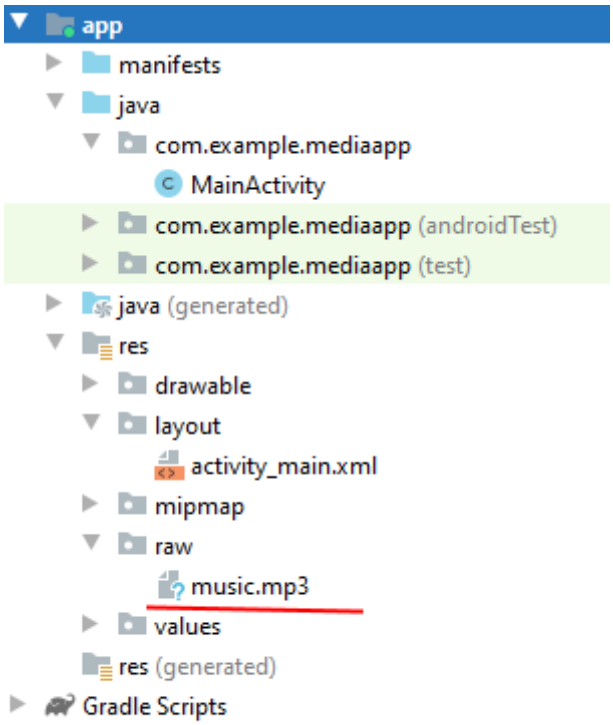
Добавление папки raw в проект Android Studio

Затем в появившемся окне в качестве типа папки укажем **raw** (что также будет использоваться в качестве названия папки):



Создание папки raw в проект Android Studio

И скопируем в нее какой-нибудь аудиофайл.



Добавление аудиофайла в Android Studio

Для управления аудиопотоком определим в файле **activity_main.xml** три кнопки:

```
<?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">
    <Button
        android:id="@+id/playButton"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Play"
        android:onClick="play"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toLeftOf="@id/pauseButton"
        app:layout_constraintTop_toTopOf="parent" />
    <Button
        android:id="@+id/pauseButton"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Pause"
        android:onClick="pause"
        app:layout_constraintLeft_toRightOf="@id/playButton"
        app:layout_constraintRight_toLeftOf="@id/stopButton"
        app:layout_constraintTop_toTopOf="parent"/>
    <Button
        android:id="@+id/stopButton"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Stop"
```

```
        android:onClick="stop"
        app:layout_constraintLeft_toRightOf="@id/pauseButton"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

И изменим код класса **MainActivity**:

```
package com.example.mediaapp;
import androidx.appcompat.app.AppCompatActivity;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {

    MediaPlayer mPlayer;
    Button playButton, pauseButton, stopButton;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mPlayer= MediaPlayer.create(this, R.raw.music);
        mPlayer.setOnCompletionListener(new MediaPlayer.OnCompletionListener()
{
            @Override
            public void onCompletion(MediaPlayer mp) {
```

```
            stopPlay();
        }
    });
    playButton = findViewById(R.id.playButton);
    pauseButton = findViewById(R.id.pauseButton);
    stopButton = findViewById(R.id.stopButton);

    pauseButton.setEnabled(false);
    stopButton.setEnabled(false);
}
private void stopPlay(){
    mPlayer.stop();
    pauseButton.setEnabled(false);
    stopButton.setEnabled(false);
    try {
        mPlayer.prepare();
        mPlayer.seekTo(0);
        playButton.setEnabled(true);
    }
    catch (Throwable t) {
        Toast.makeText(this, t.getMessage(), Toast.LENGTH_SHORT).show();
    }
}
public void play(View view){

    mPlayer.start();
    playButton.setEnabled(false);
    pauseButton.setEnabled(true);
    stopButton.setEnabled(true);
```

```
    }
    public void pause(View view){

        mPlayer.pause();
        playButton.setEnabled(true);
        pauseButton.setEnabled(false);
        stopButton.setEnabled(true);
    }
    public void stop(View view){
        stopPlay();
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        if (mPlayer.isPlaying()) {
            stopPlay();
        }
    }
}
```

Обработчик каждой кнопки кроме вызова определенного метода у MediaPlayer также переключает доступность кнопок.

И если запуск и приостановка воспроизведения особых сложностей не вызывает, то при обработки полной остановки воспроизведения мы можем столкнуться с рядом трудностей. В частности, когда мы выходим из приложения - полностью закрываем его через диспетчер приложений либо нажимаем на кнопку Назад, то у нас для текущей Activity вызывается метод onDestroy, activity уничтожается, но **MediaPlayer** продолжает работать. Если мы вернемся к приложению, то activity будет создана заново, но с помощью кнопок мы не сможем управлять воспроизведением. Поэтому в данном

случае переопределяем метод **onDestroy**, в котором завершаем воспроизведение.

Для корректного завершения также определен обработчик естественного завершения воспроизведения **OnCompletionListener**, действие которого будет аналогично нажатию на кнопку "Стоп".

Воспроизведение аудио в Android и MediaPlayer

Добавим к воспроизведению индикатор громкости. Для этого в файле **activity_main.xml** определим **SeekBar:**

```
<?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">
```

```
<Button
    android:id="@+id/playButton"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Play"
    android:onClick="play"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toLeftOf="@id/pauseButton"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toTopOf="@id/volumeControl" />
<Button
    android:id="@+id/pauseButton"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Pause"
    android:onClick="pause"
    app:layout_constraintLeft_toRightOf="@id/playButton"
    app:layout_constraintRight_toLeftOf="@id/stopButton"
    app:layout_constraintTop_toTopOf="parent"/>
<Button
    android:id="@+id/stopButton"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:text="Stop"
    android:onClick="stop"
    app:layout_constraintLeft_toRightOf="@id/pauseButton"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
<SeekBar
```

```
        android:id="@+id/volumeControl"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        app:layout_constraintTop_toBottomOf="@id/playButton"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintLeft_toLeftOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

И далее изменим код класса **MainActivity**:

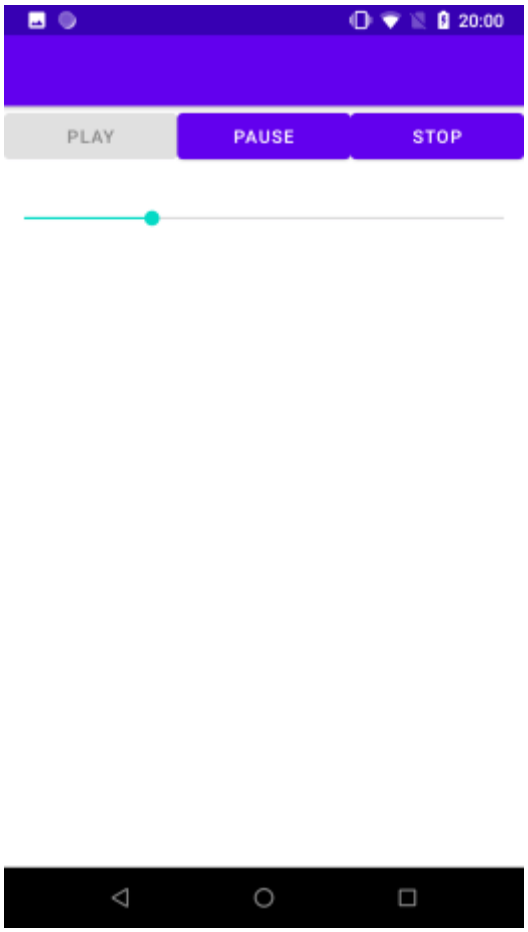
```
package com.example.mediaapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Context;
import android.media.AudioManager;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.SeekBar;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    MediaPlayer mPlayer;
    Button playButton, pauseButton, stopButton;
    SeekBar volumeControl;
    AudioManager audioManager;
```

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

        mPlayer=MediaPlayer.create(this, R.raw.music);
        mPlayer.setOnCompletionListener(new MediaPlayer.OnCompletionListener()
{
            @Override
            public void onCompletion(MediaPlayer mp) {
                stopPlay();
            }
        });
        playButton = findViewById(R.id.playButton);
        pauseButton = findViewById(R.id.pauseButton);
        stopButton = findViewById(R.id.stopButton);

        audioManager = (AudioManager)
getSystemService(Context.AUDIO_SERVICE);
        int maxVolume =
audioManager.getStreamMaxVolume(AudioManager.STREAM_MUSIC);
        int curValue =
audioManager.getStreamVolume(AudioManager.STREAM_MUSIC);

        volumeControl = findViewById(R.id.volumeControl);
        volumeControl.setMax(maxVolume);
        volumeControl.setProgress(curValue);
        volumeControl.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener() {
```

```
        @Override
        public void onProgressChanged(SeekBar seekBar, int progress, boolean
fromUser) {
            audioManager.setStreamVolume(AudioManager.STREAM_MUSIC,
progress, 0);
        }
        @Override
        public void onStartTrackingTouch(SeekBar seekBar) {

        }
        @Override
        public void onStopTrackingTouch(SeekBar seekBar) {

        }
    });

    pauseButton.setEnabled(false);
    stopButton.setEnabled(false);
  }
  private void stopPlay(){
    mPlayer.stop();
    pauseButton.setEnabled(false);
    stopButton.setEnabled(false);
    try {
      mPlayer.prepare();
      mPlayer.seekTo(0);
      playButton.setEnabled(true);
    }
    catch (Throwable t) {
```

```
            Toast.makeText(this, t.getMessage(), Toast.LENGTH_SHORT).show();
        }
    }

    public void play(View view){

        mPlayer.start();
        playButton.setEnabled(false);
        pauseButton.setEnabled(true);
        stopButton.setEnabled(true);
    }
    public void pause(View view){

        mPlayer.pause();
        playButton.setEnabled(true);
        pauseButton.setEnabled(false);
        stopButton.setEnabled(true);
    }
    public void stop(View view){
        stopPlay();
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        if (mPlayer.isPlaying()) {
            stopPlay();
        }
    }
}
```

Для управления громкостью звука применяется класс **AudioManager**. А с помощью вызова **audioManager.setStreamVolume(AudioManager.STREAM_MUSIC, progress, 0)**; в качестве второго параметра можно передать нужное значение громкости.

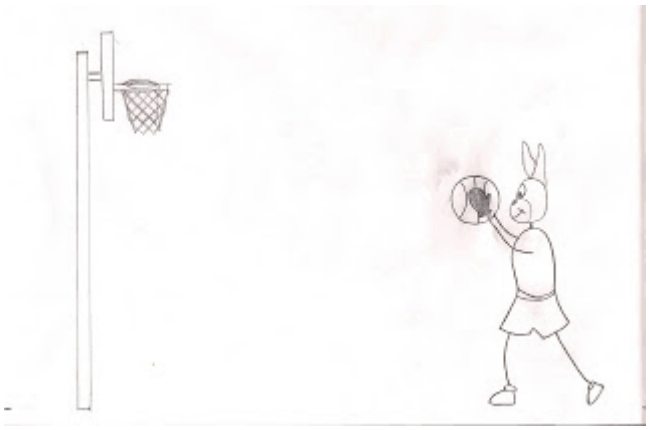


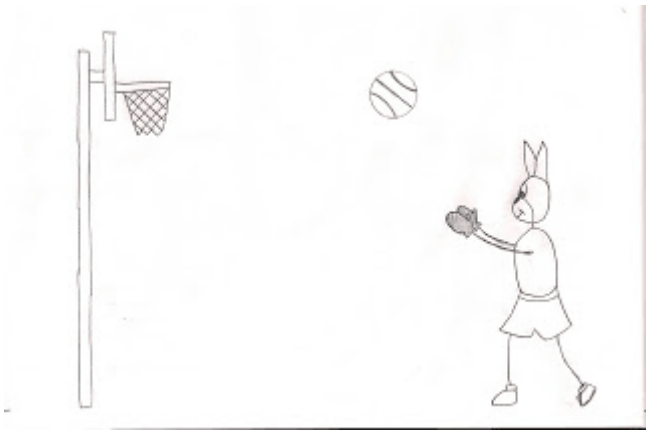
Громкость в MediaPlayer в Android и Java

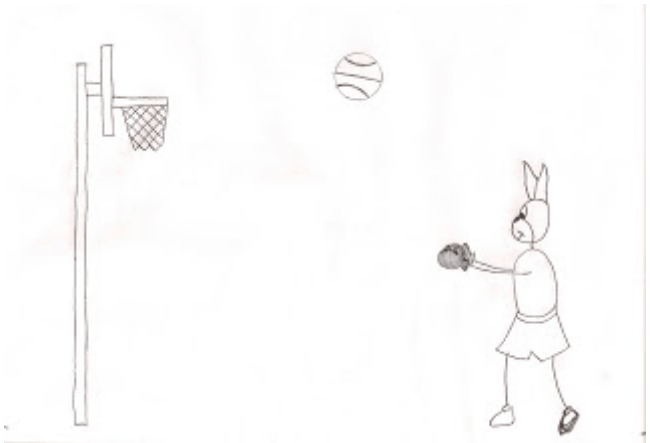
Анимация

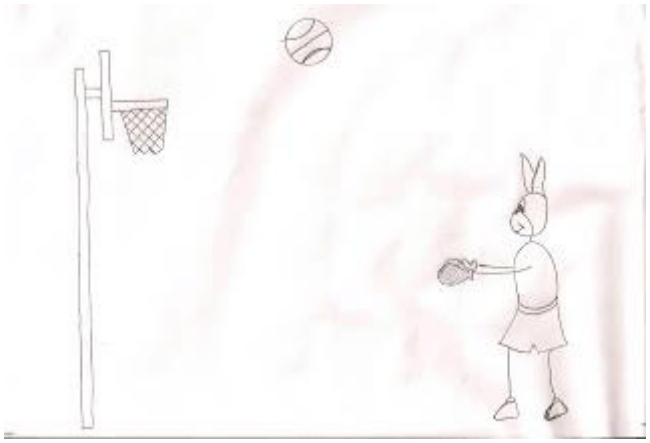
Cell-анимация

Cell animation или анимация фреймов представляет собой технику анимации, при которой ряд изображений или кадров/фреймов последовательно сменяют друг друга за короткий промежуток времени. Подобная техника довольно распространена при создании мультфильмов. Например, имеется следующий набор изображений:





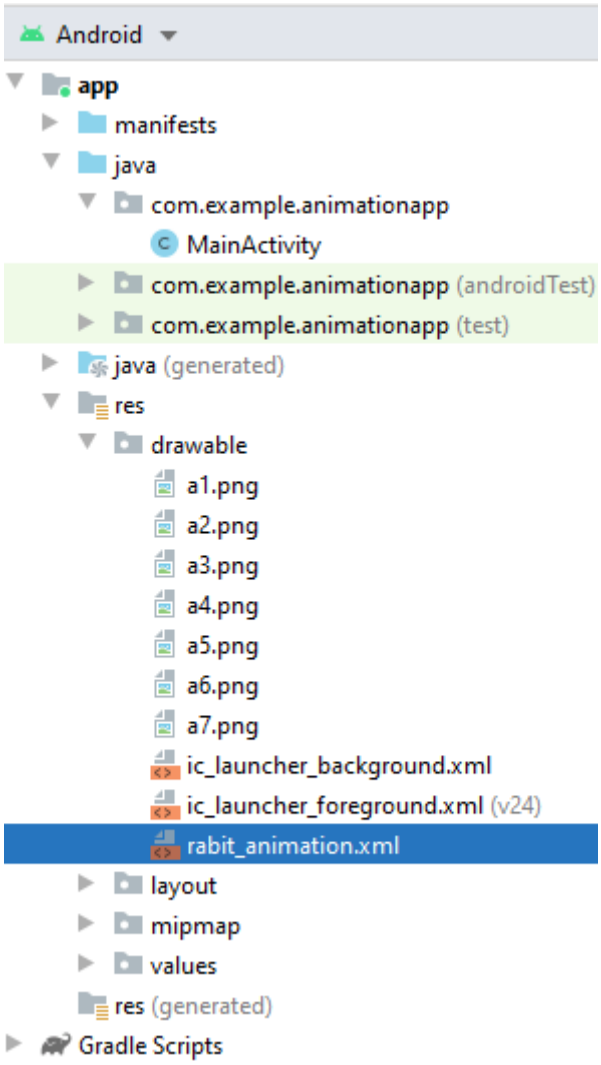




При достаточно быстрой смене кадров получится динамический эффект зайца, забрасывающего мяч в баскетбольную корзину. Теперь рассмотрим, как сделать подобную анимацию в приложении Android.

Во-первых, нам надо добавить все эти изображения в проект в папку **res/drawable**. И в эту же папку добавим новый xml-файл. Назовем его **rabit_animation.xml** и поместим в него следующее содержимое:

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false" >
    <item android:drawable="@drawable/a1" android:duration="250" />
    <item android:drawable="@drawable/a2" android:duration="250" />
    <item android:drawable="@drawable/a3" android:duration="250" />
    <item android:drawable="@drawable/a4" android:duration="250" />
    <item android:drawable="@drawable/a5" android:duration="250" />
    <item android:drawable="@drawable/a6" android:duration="250" />
    <item android:drawable="@drawable/a7" android:duration="250" />
</animation-list>
```



Анимация фреймов animation-list в Android и Java

Анимация определяется с помощью корневого элемента **animation-list**, который содержит набор ключевых кадров в виде элементов item.

Свойство **android:oneshot="false"** в определении корневого элемента указывает, что анимация будут продолжатся циклически. А при значении true анимация срабатывала только один раз.

Каждый элемент аннимации устанавливает ссылку на ресурс изображения с помощью свойства **android:drawable**, а также с помощью свойства **android:duration** устанавливает время в миллисекундах, которое будет отображаться изображение.

В разметке интерфейса для отображения анимации используется 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/animationView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:adjustViewBounds="true"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

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

Далее изменим код MainActivity, чтобы запустить анимацию:

```
package com.example.animationapp;

import androidx.appcompat.app.AppCompatActivity;

import android.graphics.drawable.AnimationDrawable;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity {

    @Override
```

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


        ImageView img = findViewById(R.id.animationView);
        // устанавливаем ресурс анимации
        img.setBackgroundResource(R.drawable.rabit_animation);
        // получаем объект анимации
        AnimationDrawable frameAnimation = (AnimationDrawable)
img.getBackground();
        // по нажатию на ImageView
        img.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // запускаем анимацию
                frameAnimation.start();
            }
        });
    }
}
```

С помощью метода **setBackgroundResource()** объекта ImageView устанавливается ресурс анимации. Затем из ImageView получаем собственно объект анимации **AnimationDrawable** и по нажатию на ImageView запускаем анимацию с помощью метода **start().**

Анимация фреймов AnimationDrawable в Android и Java

Стоит отметить, что метод **start()** объекта **AnimationDrawable** не вызывается напрямую из onCreate() класса MainActivity, так как при выполнении метода **onCreate()** объект **AnimationDrawable** еще полностью не определен. Поэтому в данном случае анимация запускается именно при нажатии на ImageIview, когда приложение видимо на экране и взаимодействует с пользователем. Если же необходимо автоматически запустить анимацию при запуске приложения, то можно это делать в методе **onStart()** класса Activity.

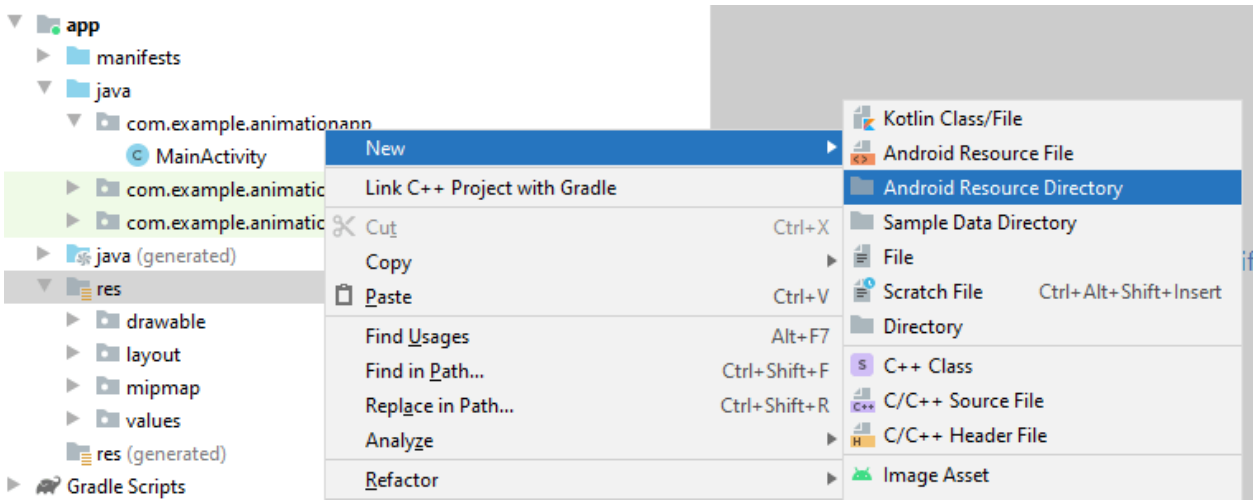
Tween-анимация

Tween-анимация представляет анимацию различных свойств объекта, при которой система сама рассчитывает некоторые промежуточные значения с помощью определенного алгоритма, который называется интерполяцией. В

Android алгоритм интерполяции определяется встроенным классом **Animation.**

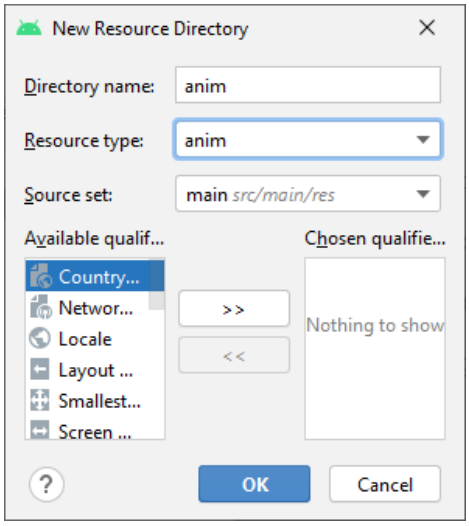
От данного класса наследуются классы, которые описывают конкретные типы анимаций, такие как **AlphaAnimation** (изменение прозрачности), **RotateAnimation** (анимация вращения**), ScaleAnimation** (анимация масштабирования), **TranslateAnimation** (анимация перемещения).

Мы можем определить анимацию как в коде java, так и в файле xml. Для хранения ресурсов анимации в папке **res** предназначена папка **anim**. По умолчанию данная папка отсутствует в проекте, поэтому создадим ее. Для этого нажмем правой кнопкой мыши на папку res и в появившемся контекстном меню выберем пункт **New -> Android Resource Directory:**



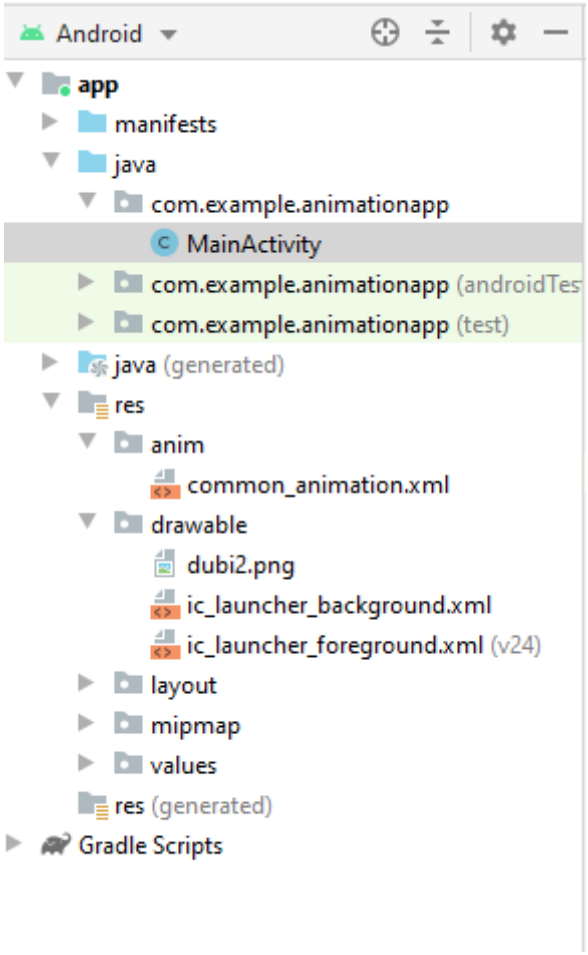
Tween-анимация в Android Studio

Затем в появившемся окне укажем тип ресурсов – **anim**



Resource anim in Android Studio

Далее добавим в нее новый xml-файл, который назовем **common_animation.xml:**



Tween-анимация в Android

Определим в этом файле следующее содержание:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/linear_interpolator">
    <scale android:fromXScale="1.0" android:toXScale="0.5"
        android:fromYScale="1.0" android:toYScale="0.5"
        android:pivotX="50%" android:pivotY="50%" android:duration="4500"
        android:repeatCount="infinite" android:repeatMode="reverse" />
    <rotate
        android:fromDegrees="0.0"
```

```
        android:toDegrees="60.0"
        android:pivotX="50%"
        android:pivotY="50%" />
    <alpha android:fromAlpha="1.0" android:toAlpha="0.1"
android:duration="2250"
        android:repeatCount="infinite" android:repeatMode="reverse" />
    <translate android:fromXDelta="0.0"
        android:toXDelta="50.0"
        android:fromYDelta="20.0"
        android:toYDelta="80.0"
        android:duration="2250"
        android:repeatMode="reverse"
        android:repeatCount="infinite" />
</set>
```

Здесь задействуются четыре типа анимаций: элемент **scale** представляет масштабирование, элемент **rotate** - вращение, элемент alpha - изменение прозрачности, элемент **translate** - перемещение. Если бы мы использовали одну анимацию, то могли бы определить один корневой элемент по типу анимации. Но так как мы используем набор, то все анимации помещаются в элемент **set**, который представляет класс **AnimationSet**

Все виды анимаций принимают ряд общих свойств. В частности, свойство **android:repeatMode**, которое указывает на режим выполнения. Если имеет значение **reverse**, то анимация выполняется также и в обратную сторону

Свойство **android:repeatCount** указывает на количество повторов анимации. Значение **infinite** устанавливает бесконечное число повторов.

Время анимации задается с помощью свойства **android:duration**

Для всех анимаций также характерно указание начальной и конечной точки трансформации.

Анимация масштабирования

Для анимация масштабирования задается начальное масштабирование по оси x **(android:fromXScale)** и по оси y **(android:fromYScale)** и конечные значения масштабирования **android:toXScale** и **android:toYScale**. Например, так как **android:fromXScale=1.0**, а **android:toXScale=0.5**, то по ширине будет происходить сжатие на 50%.

Также при масштабировании устанавливаются зачения android:pivotX и android:pivotY, которые указывают на центр масшабирования или опорную точку.

Анимация вращения

Для анимации вращения задается начальное **(android:fromDegrees)** и конечное значения поворота **(android:toDegrees)**.

С помощью свойств android:pivotX и android:pivotY также, как и при масштабировании, задается опорная точка вращения.

Анимация прозрачности

Для анимации прозрачности задается начальное значение прозрачности **(android:fromAlpha)** и финальное значение, устанавливамое при завершении анимации **(android:toAlpha)**. Все значения варьируют в диапазоне от 1.0 (непрозрачный) до 0.0 (полностью прозрачный)

Анимация перемещения

Для перемещения также устанавливаются начальные **(android:fromXDelta** и **android:fromYDelta)** и конечные значения **(android:toXDelta** и **android:toYDelta)**

Для всех анимаций начальные и конечные значения указывают некий диапазон, в котором будут ранжироваться значения. Само вычисление значений на этом промежутке зависит от конкретного алгоритма. В данном случае в качестве алгоритма устанавливается линейная интерполяция. Для этого у корневого элемента **set** определено свойство **android:interpolator="@android:anim/linear_interpolator"**.

Кроме данного значения свойство **android:interpolator** может принимать еще ряд других: bounce_interpolator, cycle_interpolator и т.д.

Данная анимация будет применяться к элементу **ImageView,** который определим в файле разметки интерфейса:

```
<?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/animationView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:adjustViewBounds="true"

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

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

Для демонстрации анимации добавим в папку **res/drawable** какой-нибудь графический файл. В моем случае это файл **dubi2.png**.

Теперь определим и запустим анимацию в классе **MainActivity**:

```
package com.example.animationapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity {

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

        ImageView img = findViewById(R.id.animationView);
        // определим для ImageView какое-нибудь изображение
        img.setImageResource(R.drawable.dubi2);
        // создаем анимацию
        Animation animation = AnimationUtils.loadAnimation(this,
R.anim.common_animation);
        // запуск анимации
        img.startAnimation(animation);
    }
```

}

Сначала определяем анимацию по тому файлу common_animation.xml, который содержит набор анимаций:

```
Animation animation = AnimationUtils.loadAnimation(this,
R.anim.common_animation);
```

А потом запускаем ее:

```
img.startAnimation(animation);
```

Анимации в Android и Java

Сервисы.

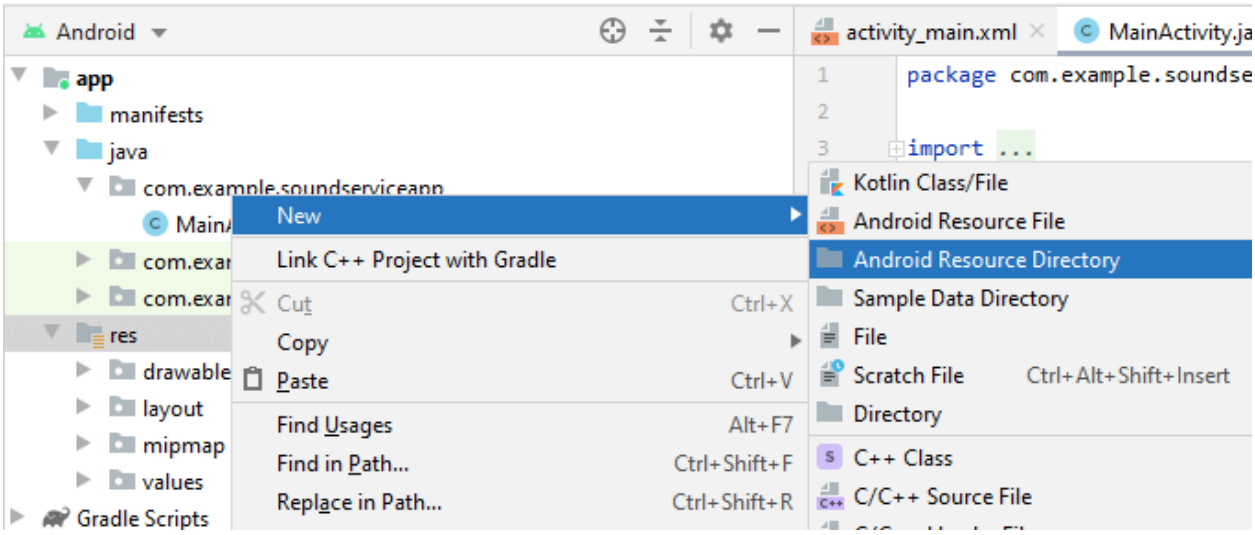
Введение в сервисы Android

Сервисы представляют собой особую организацию приложения. В отличие от activity они не требуют наличия визуального интерфейса. Сервисы позволяют выполнять долговременные задачи без вмешательства пользователя.

Все сервисы наследуются от класса **Service** и проходят следующие этапы жизненного цикла:

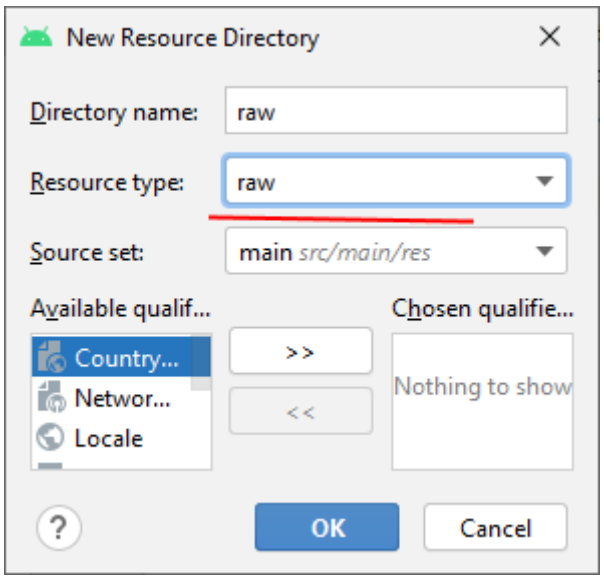
- Метод onCreate(): вызывается при создании сервиса
- Метод onStartCommand(): вызывается при получении сервисом команды, отправленной с помощью метода startService()
- Метод onBind(): вызывается при закреплении клиента за сервисом с помощью метода bindService()
- Метод onDestroy(): вызывается при завершении работы сервиса

Создадим простейшее приложение с сервисом. Наш сервис будет воспроизводить музыкальный файл. И вначале добавим в проект в каталог res папку **raw**. Для этого нажмем правой кнопкой мыши на каталог res и в контекстном меню выберем пункт **New -> Android Resource Directory**.



Добавление папки raw для сервисов в Android Studio и Java

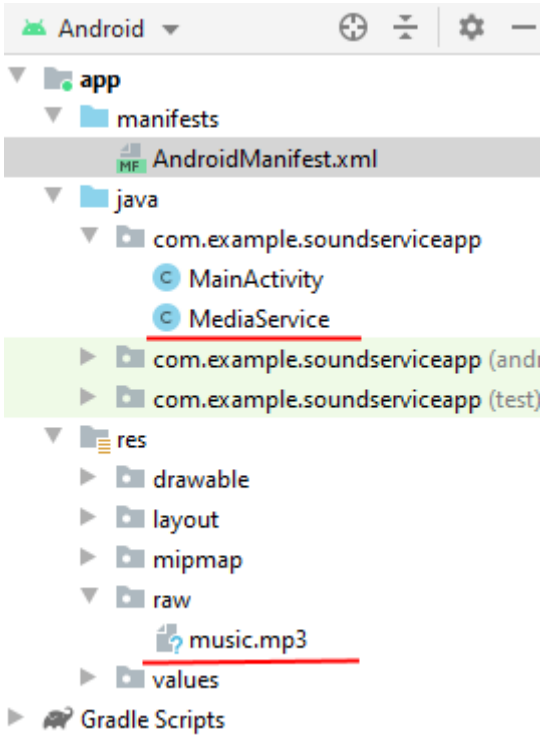
Далее укажем в качестве типа папки - **raw**:



Добавление папки ресурсов raw для сервисов в Android Studio и Java

И поместим в эту папку (**res/raw**) какой-нибудь mp3-файл.

Затем добавим новый класс сервиса. Назовем его **MediaService**. В итоге получится следующий проект:



Добавление сервисов в Android Studio и Java

Для воспроизведения аудио-файла определим в классе **MediaService** следующий код:

```
package com.example.soundserviceapp;

import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;

public class MediaService extends Service {
    MediaPlayer ambientMediaPlayer;
    @Override
    public IBinder onBind(Intent intent) {
```

```
        throw new UnsupportedOperationException("Not yet implemented");
    }
    @Override
    public void onCreate(){
        ambientMediaPlayer=MediaPlayer.create(this, R.raw.music);
        ambientMediaPlayer.setLooping(true);
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId){
        ambientMediaPlayer.start();
        return START_STICKY;
    }
    @Override
    public void onDestroy() {
        ambientMediaPlayer.stop();
    }
}
```

Для воспроизведения музыкального файла сервис будет использовать компонент **MediaPlayer.**

В сервисе переопределяются все четыре метода жизненного цикла. Но по сути метод **onBind()** не имеет никакой реализации.

В методе **onCreate()** инициализируется медиа-проигрыватель с помощью музыкального ресурса, который добавлен в папку res/raw.

В методе **onStartCommand()** начинается воспроизведение.

Метод **onStartCommand()** может возвращать одно из значений, которое предполагает различное поведение в случае, если процесс сервиса был неожиданно завершен системой:

- **START_STICKY**: в этом случае сервис снова возвращается в запущенное состояние, как будто если бы снова был бы вызван метод onStartCommand() без передачи в этот метод объекта Intent

- **START_REDELIVER_INTENT**: в этом случае сервис снова возвращается в запущенное состояние, как будто если бы снова был бы вызван метод onStartCommand() с передачей в этот метод объекта Intent

- **START_NOT_STICKY**: сервис остается в остановленном положении

Метод onDestroy() завершает воспроизведение.

Чтобы управлять сервисом, изменим activity. Сначала добавим в файл **activity_main.xml** пару кнопок для управления сервисом:

```
<?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">

    <Button
        android:id="@+id/start"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
```

```
        android:text="Старт"
        android:onClick="click"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toLeftOf="@id/stop"
        app:layout_constraintTop_toTopOf="parent" />
    <Button
        android:id="@+id/stop"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Стоп"
        android:onClick="click"
        app:layout_constraintLeft_toRightOf="@id/start"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

И изменим код **MainActivity**:

```
package com.example.soundserviceapp;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    @Override
```

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

    public void click(View v) {
        Intent i=new Intent(this, MediaService.class);
        if (v.getId()==R.id.start) {
            startService(i);
        }
        else {
            stopService(i);
        }
    }
}
```

Для запуска сервиса используется объект Intent:

```
Intent i=new Intent(this, MediaService.class);
```

Для запуска сервиса в классе **Activity** определен метод **startService()**, в который передается объект **Intent.** Этот метод будет посылать команду сервису и вызывать его метод **onStartCommand()**, а также указывать системе, что сервис должен продолжать работать до тех пор, пока не будет вызван метод stopService().

Метод **stopService()** также определен к классе Activity и принимает объект Intent. Он останавливает работу сервиса, вызывая его метод onDestroy()

И в конце нам надо зарегистрировать сервис в файле манифеста:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.soundserviceapp">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.SoundServiceApp">

        <service
            android:name=".MediaService"
            android:enabled="true"
            android:exported="true" >
        </service>

        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Регистрация сервиса производится в узле application с помощью добавления элемента <service>. В нем определяется атрибут android:name, который хранит название класса сервиса. И кроме того может принимать еще ряд атрибутов:

- android:enabled: если имеет значение "true", то сервис может ли создаваться системой. Значение по умолчанию - "true".

- android:exported: указывает, могут ли компоненты других приложений обращаться к сервису. Если имеет значение "true", то могут, если имеет значение "false", то нет.

- android:icon: значок сервиса, представляет собой ссылку на ресурс drawable

- android:isolatedProcess: если имеет значение true, то сервис может быть запущен как специальный процесс, изолированный от остальной системы.

- android:label: название сервиса, которое отображается пользователю

- android:permission: набор разрешений, которые должно применять приложение для запуска сервиса

- android:process: название процесса, в котором запущен сервис. Как правило, имеет то же название, что и пакет приложения.

Запустим приложение и нажмем на кнопку запуска сервиса:

Сервисы startService в Android и Java

После этого начнется воспроизведение добавленной нами в приложение мелодии.

Диалоговые окна

DatePickerDialog и TimePickerDialog

По умолчанию в Android уже определены два диалоговых окна - **DatePickerDialog** и **TimePickerDialog**, которые позволяют выбрать дату и время.

Кроме установки даты DatePickerDialog позволяет обработать выбор даты с помощью слушателей **OnDateChangedListener** и **OnDateSetListener**. Что позволяет использовать выбранную дату далее в приложении.

Подобным образом TimePickerDialog позволяет обработать выбор времени с помощью слушателей **OnTimeChangedListener** и **OnTimeSetListener**

Пи работе с данными компонентами надо учитывать, что отсчет месяцев в DatePickerDialog начинается с нуля, то есть январь будет иметь номер 0, а декабрь - 11. И аналогично в TimePickerDialog отсчет секунд и минут будет идти с 0 до 59, а часов - с 0 до 23.

Используем DatePickerDialog и TimePickerDialog в приложении. Определим следующую разметку интерфейса в **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/currentDateTime"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:textSize="20sp"
        app:layout_constraintBottom_toTopOf="@id/timeButton"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/timeButton"
```

```
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Изменить время"
        android:onClick="setTime"
        app:layout_constraintBottom_toTopOf="@id/dateButton"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/currentDateTime"  />

    <Button
        android:id="@+id/dateButton"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Изменить дату"
        android:onClick="setDate"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/timeButton" />

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

Здесь определены две кнопки для выбора даты и времени и текстовое поле, отображающее выбранные дату и время. И изменим код **MainActivity**:

```
package com.example.dialogsapp;

import androidx.appcompat.app.AppCompatActivity;

import android.app.DatePickerDialog;
import android.app.TimePickerDialog;
```

```
import android.os.Bundle;
import android.text.format.DateUtils;
import android.view.View;
import android.widget.DatePicker;
import android.widget.TextView;
import android.widget.TimePicker;

import java.util.Calendar;

public class MainActivity extends AppCompatActivity {

    TextView currentDateTime;
    Calendar dateAndTime=Calendar.getInstance();
    @Override
    public void onCreate(Bundle savedInstance) {
        super.onCreate(savedInstance);
        setContentView(R.layout.activity_main);
        currentDateTime = findViewById(R.id.currentDateTime);
        setInitialDateTime();
    }

    // отображаем диалоговое окно для выбора даты
    public void setDate(View v) {
        new DatePickerDialog(MainActivity.this, d,
                dateAndTime.get(Calendar.YEAR),
                dateAndTime.get(Calendar.MONTH),
                dateAndTime.get(Calendar.DAY_OF_MONTH))
                .show();
    }
```

```
    // отображаем диалоговое окно для выбора времени
    public void setTime(View v) {
        new TimePickerDialog(MainActivity.this, t,
                dateAndTime.get(Calendar.HOUR_OF_DAY),
                dateAndTime.get(Calendar.MINUTE), true)
                .show();
    }
    // установка начальных даты и времени
    private void setInitialDateTime() {

        currentDateTime.setText(DateUtils.formatDateTime(this,
                dateAndTime.getTimeInMillis(),
                DateUtils.FORMAT_SHOW_DATE |
DateUtils.FORMAT_SHOW_YEAR
                        | DateUtils.FORMAT_SHOW_TIME));
    }

    // установка обработчика выбора времени
    TimePickerDialog.OnTimeSetListener t=new
TimePickerDialog.OnTimeSetListener() {
        public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
            dateAndTime.set(Calendar.HOUR_OF_DAY, hourOfDay);
            dateAndTime.set(Calendar.MINUTE, minute);
            setInitialDateTime();
        }
    };

    // установка обработчика выбора даты
```

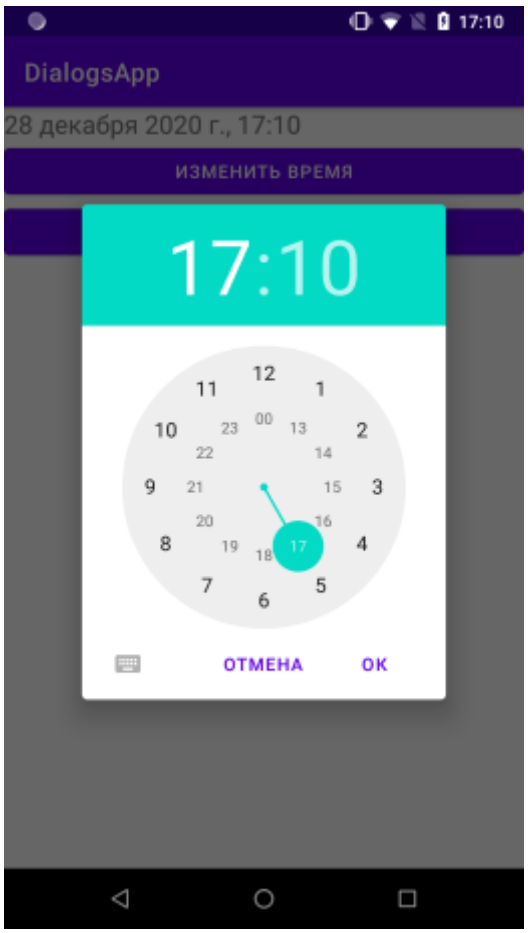
```
   DatePickerDialog.OnDateSetListener d=new
DatePickerDialog.OnDateSetListener() {
      public void onDateSet(DatePicker view, int year, int monthOfYear, int
dayOfMonth) {
         dateAndTime.set(Calendar.YEAR, year);
         dateAndTime.set(Calendar.MONTH, monthOfYear);
         dateAndTime.set(Calendar.DAY_OF_MONTH, dayOfMonth);
         setInitialDateTime();
      }
   };
}
```

Ключевым классом здесь является **java.util.Calendar**, который хранится в стандартной библиоетке классов Java. В методе **setInitialDateTime()** мы получаем из экземпляра этого класса количество миллисекунд **dateAndTime.getTimeInMillis()** и с помощью форматирования выводим на текстовое поле.

Метод **setDate()**, вызываемый по нажатию на кнопку, отображает окно для выбора даты. При создании окна его объекту передается обработчик выбора даты **DatePickerDialog.OnDateSetListener**, который изменяет дату на текстовом поле.

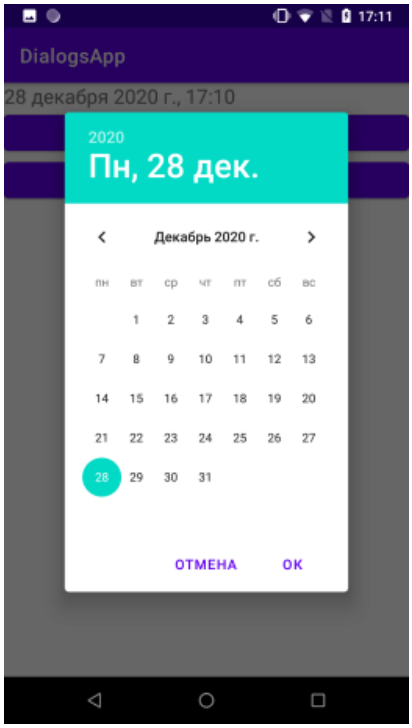
Аналогично метод setTime() отображает окно для выбора времени. Объект окна использует обработчик выбора времени **TimePickerDialog.OnTimeSetListener**, который изменяет время на текстовом поле.

И поле запуска, нажав на кнопку изменения времени, мы сможем установить время:



TimePickerDialog in Android

Аналогично работает окно установки даты:



DatePickerDialog in Android

DialogFragment и создание своих диалоговых окон

Для создания своих диалоговых окон используется компонент **AlertDialog** в связке с классом фрагмента **DialogFragment**. Рассмотрим их применение.

Вначале добавим в проект новый класс фрагмента, который назовем **CustomDialogFragment**:

```
package com.example.dialogsapp;


import android.app.AlertDialog;
import android.app.Dialog;
import android.os.Bundle;
import androidx.fragment.app.DialogFragment;
import androidx.annotation.NonNull;


public class CustomDialogFragment extends DialogFragment {


    @NonNull
    public Dialog onCreateDialog(Bundle savedInstanceState) {


        AlertDialog.Builder builder=new AlertDialog.Builder(getActivity());
        return builder.setTitle("Диалоговое окно").setMessage("Для закрытия окна нажмите ОК").create();
    }
}
```

Класс фрагмента содержит всю стандартную функциональность фрагмента с его жизненным циклом, но при этом наследуется от класса **DialogFragment**, который добавляет ряд дополнительных функций. И для его создания мы можем использовать два способа:

- Переопределение метода onCreateDialog(), который возвращает объект Dialog.

- Использование стандартного метода onCreateView().

Для создания диалогового окна в методе onCreateDialog() применяется класс AlertDialog.Builder. С помощью своих методов он позволяет настроить отображение диалогового окна:

- setTitle: устанавливает заголовок окна

- setView: устанавливает разметку интерфейса окна

- setIcon: устанавливает иконку окна

- setPositiveButton: устанавливает кнопку подтверждения действия

- setNeutralButton: устанавливает "нейтральную" кнопку, действие которой может отличаться от действий подтверждения или отмены

- setNegativeButton: устанавливает кнопку отмены

- setMessage: устанавливает текст диалогового окна, но при использовании setView данный метод необязателен или может рассматриваться в качестве альтернативы, если нам надо просто вывести сообщение.

- create: создает окно

В данном же случае диаговое кно просто выводит некоторое сообщение.

Для вызова этого диалогового окна в файле activity_main.xml определим кнопку:

```
<?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">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Dialog"
        android:onClick="showDialog"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"  />

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

В коде **MainActivity** определим обработчик нажатия кнопки, который будет запускать диалоговое окно:

```
package com.example.dialogsapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
```

```
public class MainActivity extends AppCompatActivity {


    @Override
    public void onCreate(Bundle savedInstance) {
        super.onCreate(savedInstance);
        setContentView(R.layout.activity_main);
    }


    public void showDialog(View v) {


        CustomDialogFragment dialog = new CustomDialogFragment();
        dialog.show(getSupportFragmentManager(), "custom");
    }
}
```

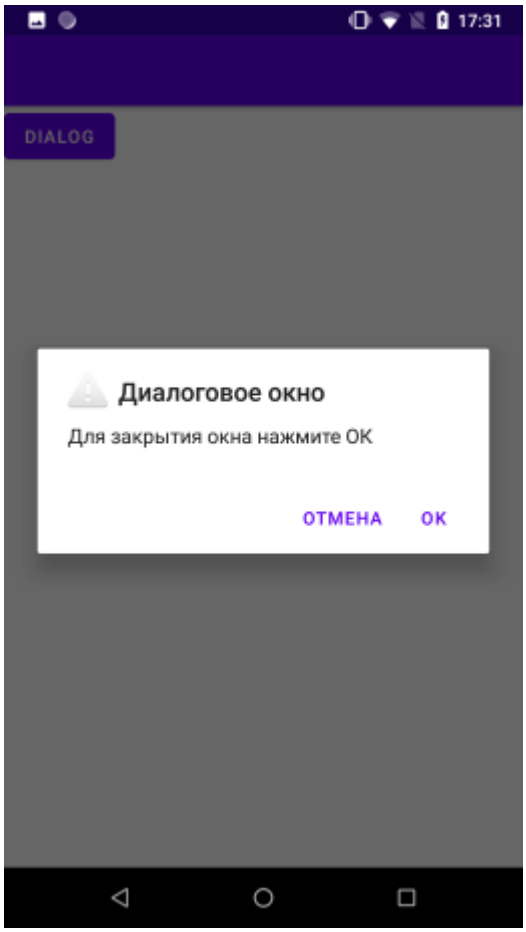
Для вызова диалогового окна создается объект фрагмента **CustomDialogFragment**, затем у него вызывается метод show(). В этот метод передается менеджер фрагментов **FragmentManager** и строка - произвольный тег.

И поле нажатия на кнопку мы сможем ввести данные в диалоговое окно:

DialogFragment и AlertDialog в Android и Java

Теперь немного кастомизируем диалоговое окно:

```
public Dialog onCreateDialog(Bundle savedInstanceState) {

    AlertDialog.Builder builder=new AlertDialog.Builder(getActivity());
    return builder
            .setTitle("Диалоговое окно")
            .setIcon(android.R.drawable.ic_dialog_alert)
            .setMessage("Для закрытия окна нажмите ОК")
            .setPositiveButton("OK", null)
            .setNegativeButton("Отмена", null)
            .create();
}
```



Диалоговые окна в Android и Java

Здесь добавляется иконка, которая в качестве изображения использует встроенный ресурс android.R.drawable.ic_dialog_alert и устанавливаются две кнопки. Для каждой кнопки можно установить текст и обработчик нажатия. В данном случае для обработчика нажатия передается null, то есть обработчик не установлен.

Теперь добавим в папку **res/layout** новый файл **dialog.xml**:

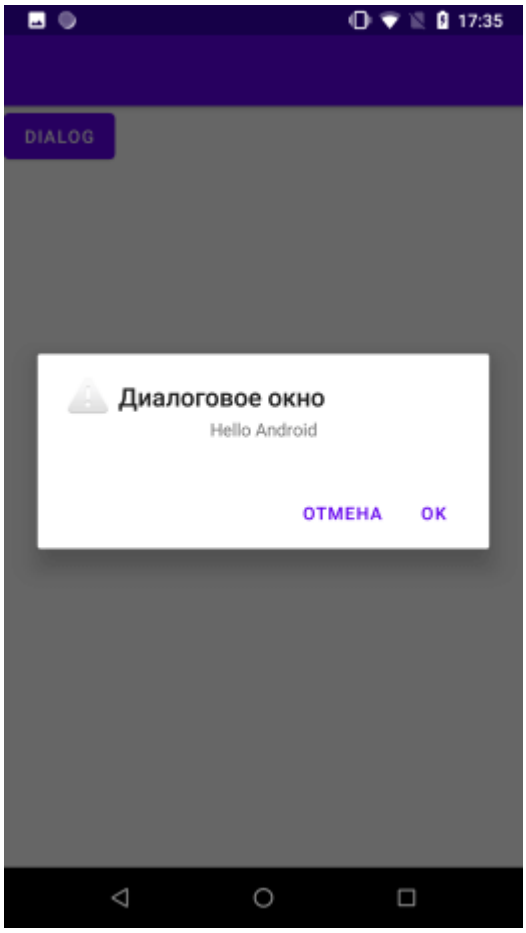
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:gravity="center"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Hello Android"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintBottom_toBottomOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

И изменим создание диалогового окна:

```
public Dialog onCreateDialog(Bundle savedInstanceState) {

    AlertDialog.Builder builder=new AlertDialog.Builder(getActivity());
    return builder
            .setTitle("Диалоговое окно")
```

```
            .setIcon(android.R.drawable.ic_dialog_alert)
            .setView(R.layout.dialog)
            .setPositiveButton("OK", null)
            .setNegativeButton("Отмена", null)
            .create();
}
```

Метод setView() устанавливает в качестве интерфейса окна ранее добавленный ресурс layout **dialog.xml**.



Создание диалоговых окон в Android

При этом, как можно увидеть на скриншоте, кнопки и заголовок с иконкой не входят в разметку.

Передача данных в диалоговое окно

Передача данных в диаговое окно, как и в любой фрагмент, осуществляется с помощью объекта Bundle.

Так, определим в файле activity_main.xml список ListView:

```
<?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">

    <ListView
        android:id="@+id/phonesList"
        android:layout_width="match_parent"
        android:layout_height="match_parent"

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

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

В классе **MainActivity** определим для этого списка данные:

```
package com.example.dialogsapp;
```

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

import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {

    @Override
    public void onCreate(Bundle savedInstance) {
        super.onCreate(savedInstance);
        setContentView(R.layout.activity_main);

        ListView phonesList = findViewById(R.id.phonesList);
        ArrayList<String> phones = new ArrayList<>();
        phones.add("Google Pixel");
        phones.add("Huawei P9");
        phones.add("LG G5");
        phones.add("Samsung Galaxy S8");

        ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_list_item_1, phones);
        phonesList.setAdapter(adapter);
```

```
        phonesList.setOnItemClickListener(new AdapterView.OnItemClickListener()
{
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position,
long id) {


                String selectedPhone = adapter.getItem(position);
                CustomDialogFragment dialog = new CustomDialogFragment();
                Bundle args = new Bundle();
                args.putString("phone", selectedPhone);
                dialog.setArguments(args);
                dialog.show(getSupportFragmentManager(), "custom");
            }
        });
    }
}
```

В обработчике нажатия на элемент в списке получаем выбранный элемент и добавляем его в объект Bundle. Далее через метод dialog.setArguments() передаем данные из Bundle во фрагмент.

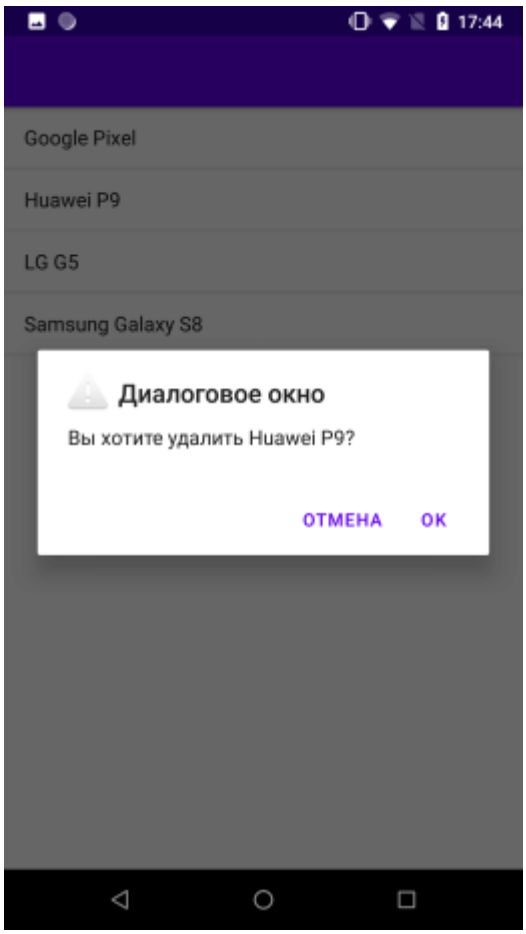
Теперь определим следующий класс фрагмента **CustomDialogFragment**:

```
package com.example.dialogsapp;


import android.app.AlertDialog;
import android.app.Dialog;
import android.os.Bundle;

import androidx.annotation.NonNull;
import androidx.fragment.app.DialogFragment;
```

```
public class CustomDialogFragment extends DialogFragment {

    @NonNull
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        String phone = getArguments().getString("phone");
        AlertDialog.Builder builder=new AlertDialog.Builder(getActivity());
        return builder
                .setTitle("Диалоговое окно")
                .setIcon(android.R.drawable.ic_dialog_alert)
                .setMessage("Вы хотите удалить " + phone + "?")
                .setPositiveButton("OK", null)
                .setNegativeButton("Отмена", null)
                .create();
    }
}
```

С помощью метода getArguments() получаем переданный в MainActivity объект Bundle. И так как была передана строка, то для ее извлечения применяется метод getString().

И при нажатии элемент списка будет передаваться в диалоговое окно:



Передача данных в диалоговое окно AlertDialog и DialogFragment в Android и Java

В данном случае реального удаления не происходит, и в следующей статье рассмотрим, как добавить логику удаления и взаимодействия с Activity

Взаимодействие диалогового окна с Activity

Взаимодействие между Activity и фрагментом производится, как правило, через интерфейс. К примеру, в прошлой теме MainActivity выводила список объектов, и теперь определим удаление из этого списка через диалоговое окно.

Для этого добавим в проект интерфейс **Removable**:

```
package com.example.dialogsapp;

public interface Removable {

    void remove(String name);

}
```

Единственный метод интерфейса remove получает удаляемый объект в виде параметра name.

Теперь реализуем этот интерфейс в коде **MainActivity**:

```
package com.example.dialogsapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity  implements  Removable{

    private ArrayAdapter<String> adapter;
    @Override
    public void onCreate(Bundle savedInstance) {
        super.onCreate(savedInstance);
        setContentView(R.layout.activity_main);

        ListView phonesList = findViewById(R.id.phonesList);
        ArrayList<String> phones = new ArrayList<>();
        phones.add("Google Pixel");
        phones.add("Huawei P9");
        phones.add("LG G5");
        phones.add("Samsung Galaxy S8");
```

```
        adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1,
phones);
        phonesList.setAdapter(adapter);


        phonesList.setOnItemClickListener(new AdapterView.OnItemClickListener()
{
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position,
long id) {


                String selectedPhone = adapter.getItem(position);
                CustomDialogFragment dialog = new CustomDialogFragment();
                Bundle args = new Bundle();
                args.putString("phone", selectedPhone);
                dialog.setArguments(args);
                dialog.show(getSupportFragmentManager(), "custom");
            }
        });
    }
    @Override
    public void remove(String name) {
        adapter.remove(name);
    }
}
```

Метод remove просто удаляет из адаптера переданный элемент.

Файл **activity_main.xml** по прежнему определяет только элемент ListView:

```
<?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">

    <ListView
        android:id="@+id/phonesList"
        android:layout_width="match_parent"
        android:layout_height="match_parent"

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

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

И в конце определим фрагмент **CustomDialogFragment**:

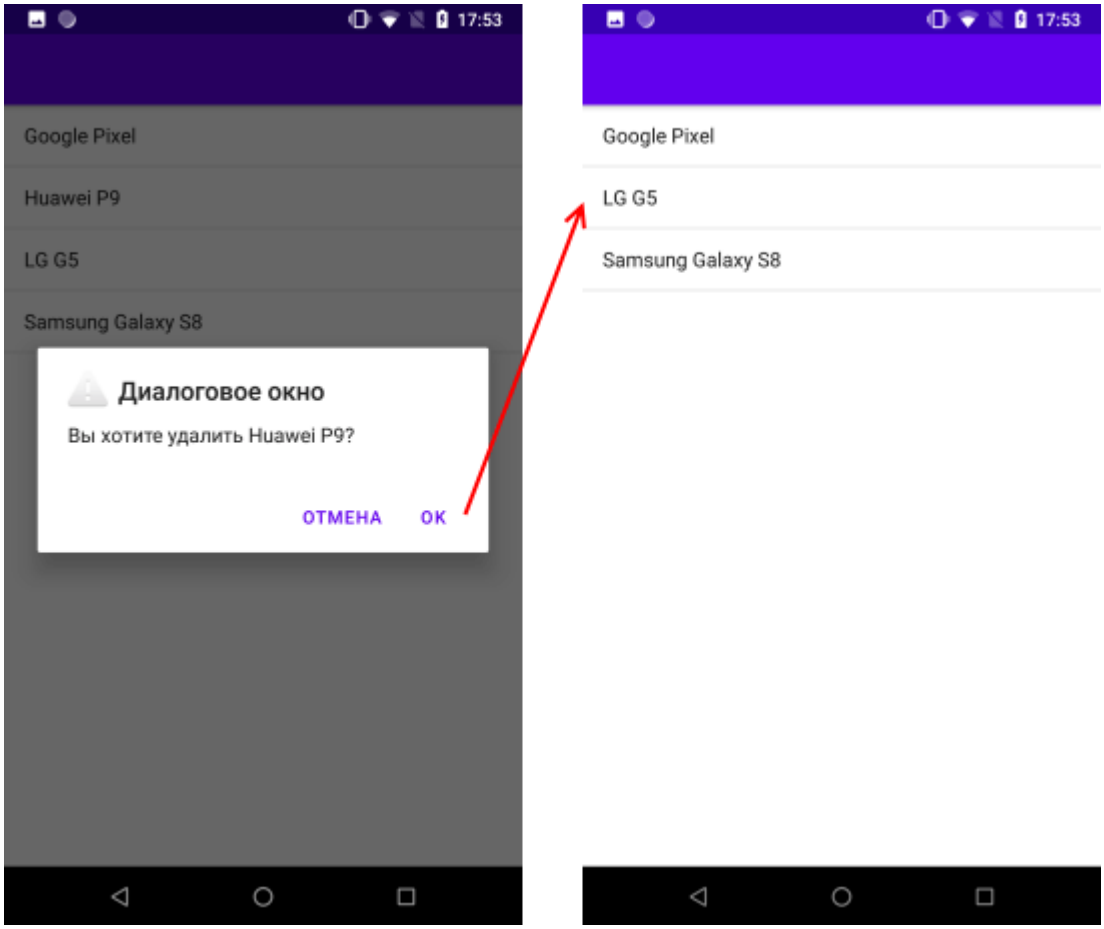
```
package com.example.dialogsapp;

import android.app.AlertDialog;
import android.app.Dialog;
import android.content.Context;
import android.content.DialogInterface;
import android.os.Bundle;

import androidx.annotation.NonNull;
import androidx.fragment.app.DialogFragment;
```

```
public class CustomDialogFragment extends DialogFragment {

    private Removable removable;

    @Override
    public void onAttach(Context context){
        super.onAttach(context);
        removable = (Removable) context;
    }

    @NonNull
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        final String phone = getArguments().getString("phone");
        AlertDialog.Builder builder=new AlertDialog.Builder(getActivity());
        return builder
                .setTitle("Диалоговое окно")
                .setIcon(android.R.drawable.ic_dialog_alert)
                .setMessage("Вы хотите удалить " + phone + "?")
                .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        removable.remove(phone);
                    }
                })
                .setNegativeButton("Отмена", null)
                .create();
    }
```

}

Метод **onAttach()** вызывается в начале жизненного цикла фрагмента, и именно здесь мы можем получить контекст фрагмента, в качестве которого выступает класс MainActivity. Так как MainActivity реализует интерфейс Removable, то мы можем преобразовать контекст к данному интерфейсу.

Затем в обработчике кнопки OK вызывается метод remove объекта Removable, который удаляет переданный во фрагмент объект phone.



Взаимодействие с MainActivity в диалоговом окне AlertDialog и DialogFragment в Android и Java

Задание

1. Реализовать приложение для работы с видео материалами в стандартном наборе виджетов применяя класс **VideoView**
2. С помощью класса MediaController добавить к VideoView дополнительно элементы управления.
3. Реализовать воспроизведение файла из интернета.
4. Реализовать приложение для работы с аудио материалами, применяя класс **MediaPlayer**
5. Реализовать приложение для работы с анимацией, применяя технологию Cell animation.
6. Реализовать приложение для работы с анимацией, применяя технологию Tween- animation. Создать приложение с сервисом со всеми необходимыми этапами жизненного цикла, применяя класс Service.
7. Реализовать диалоговые окна, применяя технологии DatePickerDialog и TimePickerDialog, которые позволяют выбрать дату и время.
8. Применяя класс DialogFragment создать свои диалоговые окна.
9. Реализовать передачу данных в диалоговое окно, как и в любой фрагмент, с помощью объекта Bundle.
10. Реализовать взаимодействие диалогового окна с Activity