**DialogFragment и создание 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.eugene.dialogsapp;

import android.app.AlertDialog;

import android.app.Dialog;

import android.os.Bundle;

import android.support.annotation.NonNull;

import android.support.v4.app.DialogFragment;

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 использвоать 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:

<LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"

    android:layout\_width="match\_parent"

    android:layout\_height="match\_parent"

    android:orientation="vertical">

    <Button

        android:layout\_width="wrap\_content"

        android:layout\_height="wrap\_content"

        android:text="Dialog"

        android:onClick="showDialog"/>

</LinearLayout>

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

package com.example.eugene.dialogsapp;

import android.support.v7.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();

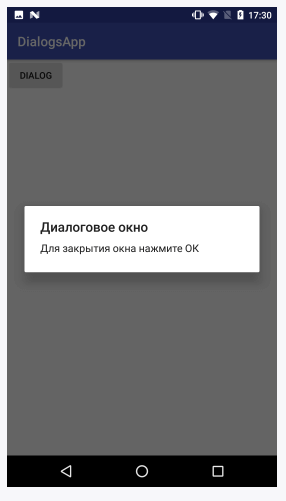
        dialog.show(getSupportFragmentManager(), "custom");

    }

}

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:



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)

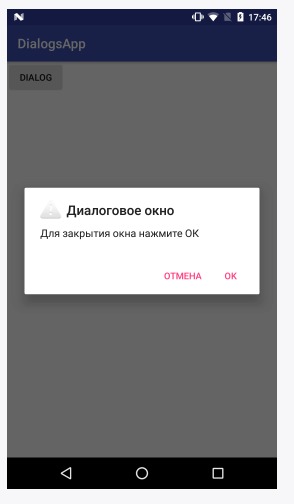
            .setMessage("Для закрытия окна нажмите ОК")

            .setPositiveButton("OK", null)

            .setNegativeButton("Отмена", null)

            .create();

}



Здесь добавляетcя 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"?>

<LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"

    android:orientation="vertical"

    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 7"/>

</LinearLayout>

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)

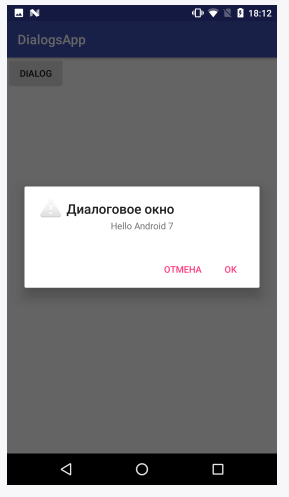
            .setPositiveButton("OK", null)

            .setNegativeButton("Отмена", null)

            .create();

}

Метод setView() устанавливает в качестве интерфейса окна ранее добавленный ресурс dialog.xml. При использовании этого метода надо учитывать, что он доступен начиная с API 21 (Lollipop). Поэтому для его применения может потребоваться изменить минимальную версию Android у проекта до 21.



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:

<LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"

    android:layout\_width="match\_parent"

    android:layout\_height="match\_parent"

    android:orientation="vertical">

    <ListView

        android:id="@+id/phonesList"

        android:layout\_width="match\_parent"

        android:layout\_height="match\_parent"  />

</LinearLayout>

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

package com.example.eugene.dialogsapp;

import android.support.v7.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 = (ListView) 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");

        final ArrayAdapter<String> adapter = new ArrayAdapter<String>(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.eugene.dialogsapp;

import android.app.AlertDialog;

import android.app.Dialog;

import android.os.Bundle;

import android.support.annotation.NonNull;

import android.support.v4.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)

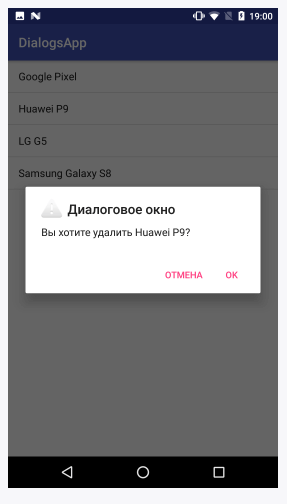
                .create();

    }

}

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:



## Взаимодействие 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 **Datable**:

package com.example.eugene.dialogsapp;

public interface Datable {

    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.eugene.dialogsapp;

import android.support.v7.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 Datable{

    private ArrayList<String> phones;

    private ArrayAdapter<String> adapter;

    @Override

    public void onCreate(Bundle savedInstance) {

        super.onCreate(savedInstance);

        setContentView(R.layout.activity\_main);

        ListView phonesList = (ListView) findViewById(R.id.phonesList);

        phones = new ArrayList<>();

        phones.add("Google Pixel");

        phones.add("Huawei P9");

        phones.add("LG G5");

        phones.add("Samsung Galaxy S8");

        adapter = new ArrayAdapter<String>(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:

<LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"

    android:layout\_width="match\_parent"

    android:layout\_height="match\_parent"

    android:orientation="vertical">

    <ListView

        android:id="@+id/phonesList"

        android:layout\_width="match\_parent"

        android:layout\_height="match\_parent"  />

</LinearLayout>

И в конце определим фрагмент **CustomDialogFragment**:

package com.example.eugene.dialogsapp;

import android.app.AlertDialog;

import android.app.Dialog;

import android.content.Context;

import android.content.DialogInterface;

import android.os.Bundle;

import android.support.annotation.NonNull;

import android.support.v4.app.DialogFragment;

public class CustomDialogFragment extends DialogFragment  {

    private Datable datable;

    @Override

    public void onAttach(Context context){

        super.onAttach(context);

        datable = (Datable) 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) {

                        datable.remove(phone);

                    }

                })

                .setNegativeButton("Отмена", null)

                .create();

    }

}

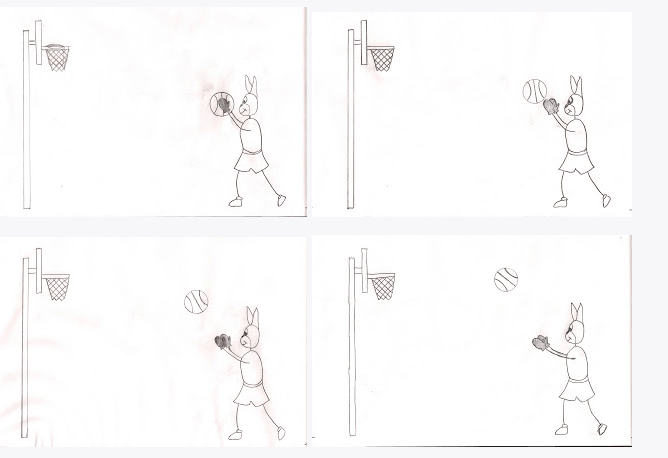
Метод onAttach() вызывается в начале жизненного цикла фрагмента, и именно здесь мы можем получить контекст фрагмента, в качестве которого выступает класс MainActivity. Так как MainActivity реализует интерфейс Datable, то мы можем преобразовать контекст к данному интерфейсу.

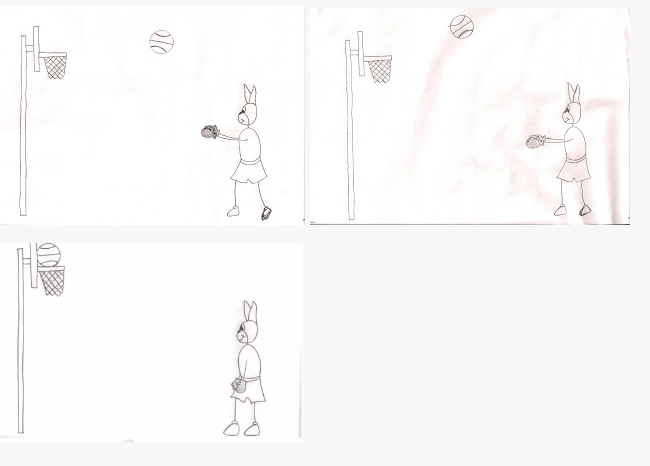
Затем в обработчике кнопки ОК вызывается метод remove объекта Datable, который удаляет переданный во фрагмент объект phone.

# Анимация

## Cell-анимация

**Cell animation** представляет собой технику анимации, при которой ряд изображений или кадров 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 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:

<RelativeLayout xmlns:android="<http://schemas.android.com/apk/res/android>"

    android:orientation="vertical"

    android:layout\_width="match\_parent"

    android:layout\_height="match\_parent">

    <ImageView android:id="@+id/animationView"

        android:layout\_width="wrap\_content"

        android:layout\_height="wrap\_content"

        android:src="@drawable/rabit\_animation"

        android:adjustViewBounds="true" android:padding="40dip"/>

</RelativeLayout>

Для установки ресурса анимации применяется свойство android:src, и также для растягивания по ширине контейнера с сохранением аспектного онтошения между шириной и высотой используется свойство android:adjustViewBounds="true"

Больше для запуска анимации ничего не надо. И при запуске приложения анимация будет автоматически запускаться.

Также можно установить анимацию в коде java:

ImageView img = (ImageView)findViewById(R.id.animationView);

img.setBackgroundResource(R.drawable.rabit\_animation);

// получаем объект анимации

AnimationDrawable frameAnimation = (AnimationDrawable) img.getBackground();

frameAnimation.setOneShot(false);

// запуск анимации

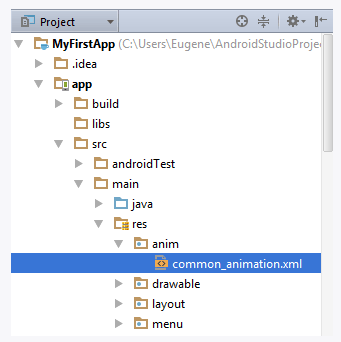
frameAnimation.start();

## Tween-анимация

Tween-анимация представляет анимацию различных свойств объекта, при которой система сама расчитывает 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 отсуствует в проекте, поэтому создадим ее. Затем добавим в нее новый xml-файл, который назовем *common\_animation.xml*:



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 редим 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 х (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 варьируютс в диапазоне от 1.0 (непрозрачный) до 0.0 (полностью прозрачный)

### Анимация перемещени

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:

<RelativeLayout xmlns:android="<http://schemas.android.com/apk/res/android>"

    android:orientation="vertical"

    android:layout\_width="match\_parent"

    android:layout\_height="match\_parent">

    <ImageView android:id="@+id/animationView"

        android:layout\_width="wrap\_content"

        android:layout\_height="wrap\_content"

        android:adjustViewBounds="true" android:padding="40dip"/>

</RelativeLayout>

Теперь определим и запустим анимацию в классе activity:

package com.example.eugene.myfirstapp;

import android.app.Activity;

import android.os.Bundle;

import android.view.Menu;

import android.view.MenuItem;

import android.widget.ImageView;

import android.view.animation.AnimationUtils;

import android.view.animation.Animation;

public class MainActivity extends Activity{

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity\_main);

        ImageView img = (ImageView)findViewById(R.id.animationView);

        // определим для ImageView какое-нибудь изображение

        img.setImageResource(R.drawable.image2);

        // 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); |

