# Анимация переходов между пунктами назначения. Transition Framework & Animation Framework



## Transition Framework & Animation Framework

В первой части этого урока мы анимируем переходы между экранами в нашем приложении. Для анимации в андроид-приложениях традиционно использовался Animation Framework.  Он позволяет анимировать view-компоненты экрана, графические ресурсы, создавать покадровые анимации. Реализовать анимацию можно как с помощью специальных xml-ресурсов, так и непосредственно в коде.

Начиная с версии Android KitKat 4.4 разработчики представили Transition Framework, который используется только в коде и позволяет буквально одной строкой описывать анимацию переходов, улучшающих пользовательский опыт. Например, вы наверняка видели в приложениях, как при нажатии на какой-либо элемент экрана он увеличивается и выходит на передний план. Такие переходы часто применяют к тексту или картинкам. Мы реализуем такие переходы во второй части урока.

## Переходы между экранами с Animation Framework

### Подготовим проект

В этом уроке будем использовать проект из задания “Создание вкладок”.

## Добавим ресурсы анимации

В папке res создайте папку anim и добавьте в нее такие файлы:

slide_in_left.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate android:fromXDelta="-50%p" android:toXDelta="0"
        android:duration="@android:integer/config_mediumAnimTime"/>
    <alpha android:fromAlpha="0.0" android:toAlpha="1.0"
        android:duration="@android:integer/config_mediumAnimTime" />
</set>
```

slide_in_right.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate android:fromXDelta="0" android:toXDelta="50%p"
        android:duration="@android:integer/config_mediumAnimTime"/>
    <alpha android:fromAlpha="1.0" android:toAlpha="0.0"
        android:duration="@android:integer/config_mediumAnimTime" />
</set>
```

slide_out_left.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate android:fromXDelta="0" android:toXDelta="-50%p"
        android:duration="@android:integer/config_mediumAnimTime"/>
    <alpha android:fromAlpha="1.0" android:toAlpha="0.0"
        android:duration="@android:integer/config_mediumAnimTime" />
</set>
```

slide_out_right.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate android:fromXDelta="0" android:toXDelta="50%p"
        android:duration="@android:integer/config_mediumAnimTime"/>
    <alpha android:fromAlpha="1.0" android:toAlpha="0.0"
        android:duration="@android:integer/config_mediumAnimTime" />
</set>
```

Как видите, во всех четырех файлах похожий код – они описывают перемещение из одной точки координат в другую с помощи директивы <translate> и изменение прозрачности от нуля до единицы при помощи <alpha>. Длительность для каждого действия можно задать в миллисекундах в свойстве duration – здесь ссылка на стандартную переменную из внутренних ресурсов android. Строго говоря, добавленные нами ресурсы тоже являются стандартными ресурсами системы android, просто взяты они из открытых исходников системы. К ним также можно обращаться и напрямую из кода, не создавая в папке res/anim вашего проекта. Однако документация не рекомендует такой подход, поскольку в разных версиях наборы ресурсов могут меняться. Поэтому рекомендуется копировать их из исходников android, доступных на гитхабе.

По ссылке вы найдете очень много различных ресурсов для анимации.

## Применим анимацию к переходу между экранами

Откройте файл графа навигации res/navigation/nav_graph.xml. На вкладке дизайна выделите стрелку перехода между экранами. В панели атрибутов справа откройте раздел Animations.

У перехода есть четыре свойства, которые можно анимировать:

- enterAnim – вход в пункт назначения
- exitAnim – выход из пункта назначения
- popEnterAnim – вход в пункт назначения с помощью pop action
- popExitAnim – выход из пункта назначения с помощью pop action

*Pop action* – это действие, при котором экраны – пункты назначения – как бы выталкиваются из стека переходов назад в процессе навигации между экранами.

Для тех, кто не знает: стек переходов назад, или back stack – это своего рода история переходов по экранам, из которой их можно открыть путем нажатия кнопки «Назад» на устройстве.

В панели атрибутов перехода в разделе Animations напротив каждого свойства нажмите кнопку «Pick a resource» и выберите созданную нами ранее анимацию для каждого свойства.

В графе навигации вы должны получить такой код в секции action первого фрагмента:

```
 <action
    android:id="@+id/action_fragmentOne_to_fragmentTwo"
    app:destination="@id/fragmentTwo"
    app:enterAnim="@anim/slide_in_right"
    app:exitAnim="@anim/slide_out_left"
    app:popEnterAnim="@anim/slide_in_left"
    app:popExitAnim="@anim/slide_out_right"/>
```

Запустите приложение и выполните переход с одного экрана на другой, а затем обратно. Как видите, все переходы сопровождаются плавной анимацией, которая делает поведение приложения более логичным. Улучшает, как говорится, пользовательский опыт.

На досуге поработайте с анимациями – поменяйте их местами, выберите другие анимации или создайте свои собственные.

А мы движемся дальше и перейдем к анимированию вьюшек при помощи Transition Framework.

## Анимация view-компонентов с Transition Framework

Изменим проект, удалив анимации перехода из action первого фрагмента в графе навигации.

Также удалите текстовое поле из макета первого фрагмента, оставив только поле ввода и кнопку.

В EditText добавьте атрибут transitionName с произвольным текстовым идентификатором, например, «editText», и задайте размер текста побольше:

```
<EditText
    android:id="@+id/editText"
    android:transitionName="editText"
    android:textSize="48sp"
…
   />
```

А в корневой ConstraintLayout добавьте фоновый цвет:

```
android:background="@color/purple_200"
```

Аналогичный атрибут transitionName с тем же значением добавьте текстовому полю второго фрагмента, и установите такой же размер текста, как в поле ввода первого фрагмента. Также удалите привязку текстового поля к родителю, чтобы расположить его слева вверху. А корневому ConstraintLayout добавьте такой фоновый цвет:

```
android:background="@color/teal_200"
```

В коде класса FragmentOne в обработчике нажатия кнопки добавьте такой код:

```
val extras = FragmentNavigatorExtras(
    editText to "editText"
)

findNavController().navigate(action, extras)
```

Здесь мы используем функцию FragmentNavigatorExtras, которая принимает пару элементов по их transitionName. Мы передаем созданную таким образом переменную extras в метод navigate контроллера навигации следующим параметром после action.

Далее нужно изменить код вывода текста, чтобы выводился только тот текст, который передается из поля ввода первого фрагмента:

```
imageButton.setOnClickListener {
    val name = editText.text.toString()
    //val hello = "Привет, $name"

    val action = FragmentOneDirections.actionFragmentOneToFragmentTwo(name)

       val extras = FragmentNavigatorExtras(            editText to "editTex
t"       )

    findNavController().navigate(action, extras)
}
```

Теперь откройте класс FragmentTwo. Переопределите метод onCreate и пропишите в нем такой код:

```
sharedElementEnterTransition = TransitionInflater.from(context).inflateTransition
(android.R.transition.move)
```

Здесь мы создаем анимацию при помощи класса TransitionInflater, передавая ему идентификатор стандартной анимации – android.R.transition.move. Вместо move можно передать другие анимации, можете попробовать самостоятельно.

Вот так, с минимальным количеством кода, мы реализовали эффектный переход между полем ввода и текстовым полем.

Чтобы посмотреть на результат, запустите приложение на устройстве. Введите на первом экране какой-нибудь текст и нажмите кнопку. Текст передается на второй экран, при этом поднимаясь вверх. Создается впечатление, что текст буквально перелетает с одного экрана на другой.