ПРАКТИЧЕСКАЯ РАБОТА 11. “АНИМАЦИЯ В ANDROID”

Внешний вид мобильного приложения играет важную роль в удобстве работы пользователей. Важно не только расположение элементов на экране, но и эффекты переходов между экранами, нажатия на кнопки, эффекты загрузки содержимого. За всё это отвечает анимация.

Android содержит различные API-интерфейсы анимации в зависимости от того, какой тип анимации необходимо использовать.

11.1. Анимация между фрагментами

Fragment API предоставляет два способа применения эффектов перехода между фрагментами во время навигации. Одним из них является Animation Framework, который использует Animation или Animator. Другой — Transition Framework, который включает в себя общие переходы элементов.

Вы можете указать пользовательские эффекты для входа и выхода из фрагментов, а также для переходов общих элементов между фрагментами.

Эффект открытия определяет, как фрагмент попадает на экран. Например, вы можете создать эффект смещения фрагмента от края экрана при переходе к нему.

Эффект закрытия определяет, как фрагмент покидает экран. Например, вы можете создать эффект постепенного исчезновения фрагмента при удалении от него.

11.1. Установка анимации между фрагментами

1. Создание анимации входа и выхода.

Необходимо в каталоге res/anim определить файлы анимации, которые будет описывать как фрагменты должны вращаться, растягиваться, исчезать и двигаться во время анимации. Пример выцветания:

```
<!-- res/anim/fade_out.xml -->
<?xml version="1.0" encoding="utf-8"?>
<alpha xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="@android:integer/config_shortAnimTime"
    android:interpolator="@android:anim/decelerate_interpolator"
    android:fromAlpha="1"
    android:toAlpha="0" />
```

Пример сдвига:

```
<!-- res/anim/slide_in.xml -->
<?xml version="1.0" encoding="utf-8"?>
<translate xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="@android:integer/config_shortAnimTime"
    android:interpolator="@android:anim/decelerate_interpolator"
    android:fromXDelta="100%"
    android:toXDelta="0%" />
```

Вы также можете указать анимацию для эффектов входа и выхода, которые запускаются при извлечении backstack'a, что может произойти, когда пользователь нажимает кнопку «Вверх» или «Назад». Они называются анимациями popEnter и popExit. Например, когда пользователь возвращается к предыдущему экрану, вы можете захотеть, чтобы текущий фрагмент соскальзывал с правого края экрана, а предыдущий фрагмент постепенно появлялся.

Эти анимации можно определить следующим образом:

```
<!-- res/anim/slide_out.xml -->
<translate xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="@android:integer/config_shortAnimTime"
    android:interpolator="@android:anim/decelerate_interpolator"
    android:fromXDelta="0%"
    android:toXDelta="100%" />
```

```
<!-- res/anim/fade_in.xml -->
<alpha xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="@android:integer/config_shortAnimTime"
    android:interpolator="@android:anim/decelerate_interpolator"
    android:fromAlpha="0"
    android:toAlpha="1" />
```

В свою очередь пример использования вышеописанных анимаций будет выглядеть следующим образом:

```
getSupportFragmentManager().beginTransaction()
    .setCustomAnimations(
        R.anim.slide_in,  // enter
        R.anim.fade_out,  // exit
        R.anim.fade_in,   // popEnter
        R.anim.slide_out  // popExit
    )
```

```
    .replace(R.id.fragment_container, fragment)
    .addToBackStack(null)
    .commit();
```

11.1.2. Установка переходов

Вы также можете использовать переходы для определения эффектов входа и выхода. Эти переходы могут быть определены в файлах ресурсов XML. Например, вы можете захотеть, чтобы текущий фрагмент исчезал, а новый фрагмент скользил от правого края экрана. Эти переходы можно определить следующим образом:

```
<!-- res/transition/fade.xml -->
<fade xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="@android:integer/config_shortAnimTime"/>
```

```
<!-- res/transition/slide_right.xml -->
<slide xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="@android:integer/config_shortAnimTime"
    android:slideEdge="right" />
```

После того, как вы определили свои переходы, примените их, вызвав setEnterTransition() для входящего фрагмента и setExitTransition() для исходящего фрагмента, передав определенные выше ресурсы перехода по их идентификатору, как показано в следующем примере:

```
public class FragmentA extends Fragment {
    @Override
    public View onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TransitionInflater inflater =
TransitionInflater.from(requireContext());
        setExitTransition(inflater.inflateTransition(R.transition.fade
));
    }
}

public class FragmentB extends Fragment {
    @Override
    public View onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TransitionInflater inflater =
TransitionInflater.from(requireContext());
        setEnterTransition(inflater.inflateTransition(R.transition.sli
```

```
de_right));
    }
}
```

11.2. Анимация растровых изображений

В некоторых ситуациях изображения необходимо анимировать. Это полезно, если вы хотите отобразить пользовательскую анимацию загрузки, состоящую из нескольких изображений, или если вы хотите, чтобы значок трансформировался после действий пользователя. Android предоставляет два способа анимации изображений.

Первый способ — использовать AnimationDrawable. Он позволяет указать несколько статических drawable файлов, которые отображаются последовательно для создания анимации. Второй вариант — использовать AnimatedVectorDrawable, который позволяет анимировать векторный рисунок.

11.2.1. AnimationDrawable

Вы можете определить кадры анимации в своем коде с помощью API класса AnimationDrawable, но проще определить их с помощью одного XML-файла, в котором перечислены кадры, составляющие анимацию. XML-файл для этого вида анимации находится в каталоге res/drawable/ проекта Android. В этом случае в инструкциях указывается порядок и продолжительность каждого кадра анимации.

XML-файл состоит из элемента <animation-list> в качестве корневого узла и ряда дочерних узлов <item>, каждый из которых определяет фрейм — ресурс для рисования и его продолжительность. Вот пример XML-файла для анимации Drawable:

```
<animation-list
xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="true">
    <item android:drawable="@drawable/rocket_thrust1"
android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust2"
android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust3"
android:duration="200" />
</animation-list>
```

Пример использования анимации в коде:

```
AnimationDrawable rocketAnimation;

public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  ImageView rocketImage = (ImageView) findViewById(R.id.rocket_image);
  rocketImage.setBackgroundResource(R.drawable.rocket_thrust);
  rocketAnimation = (AnimationDrawable) rocketImage.getBackground();

  rocketImage.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
        rocketAnimation.start();
      }
  });
}
```

11.2.2. AnimatedVectorDrawable

Векторное изображение — это тип изображения, который масштабируется без пикселизации или размытия. Класс AnimatedVectorDrawable позволяет анимировать свойства векторного рисунка, например, поворачивать его или изменять данные пути, чтобы преобразовать его в другое изображение.

Обычно вы определяете анимированные векторные рисунки в трех файлах XML:

— вектор, который можно нарисовать с помощью элемента <vector> в res/drawable/;

— анимированный вектор, который можно нарисовать с помощью элемента <animated-vector> в res/drawable/;

— один или несколько аниматоров объектов с элементом <objectAnimator> в res/animator/.

Анимированные векторные рисунки могут анимировать атрибуты элементов <group> и <path>. Элемент <group> определяет набор путей или подгрупп, а элемент <path> определяет пути для рисования.

Когда вы определяете векторный рисунок, который хотите анимировать, используйте атрибут android:name, чтобы назначить уникальное имя группам и путям, чтобы вы могли ссылаться на них из определений вашего аниматора.

Пример определения элемента <vector>:

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:height="64dp"
    android:width="64dp"
    android:viewportHeight="600"
    android:viewportWidth="600">
    <group
        android:name="rotationGroup"
        android:pivotX="300.0"
        android:pivotY="300.0"
        android:rotation="45.0" >
        <path
            android:name="v"
            android:fillColor="#000000"
            android:pathData="M300,70 l 0,-70 70,70 0,0 -70,70z" />
    </group>
</vector>
```

Пример определения элемента <animated-vector>:

```
<animated-vector
xmlns:android="http://schemas.android.com/apk/res/android"
  android:drawable="@drawable/vectordrawable" >
    <target
        android:name="rotationGroup"
        android:animation="@animator/rotation" />
    <target
        android:name="v"
        android:animation="@animator/path_morph" />
</animated-vector>
```

11.3. Анимация изменения видимости и изменения положения элементов пользовательского интерфейса

В некоторых ситуациях необходимо изменить видимость элемента пользовательского интерфейса. Для улучшения восприятия пользователем изменения видимости элемента применяют анимацию. Кроме того, могут быть ситуации, в которых необходимо изменить положения элемента на экране, хорошим тоном считается использование анимации перемещения.

Для реализации анимации перемещения элемента или изменения его видимости создан пакет android.animation. Он доступен начиная с API 11. С его помощью можно изменять свойства элемента, представленного классом View и, таким образом, создавать анимацию. Одним из таких примеров является изменение свойства alpha, благодаря которому можно плавно изменять видимость элемента.

Применение анимации к элементу View осуществляется с помощью метода animate, представленного в классе View. Для изменения видимости используется вызов метода alpha. Настройка скорости изменения видимости осуществляется с помощью изменения длительности анимации. Длительность анимации может быть указана с помощью метода setDuration. В некоторых случаях, необходимо выполнить код после завершения анимации, для этого необходимо указать listener окончания анимации с помощью метода setListener. Используется интерфейс слушателя AnimationListenerAdapter, в котором переопределяется метод onAnimationEnd. Таким обраом, после завершения анимации будет вызван метод слушателя onAnimationEnd и выполнены требуемые действия.

Пример создания анимации затухания для изменения видимости элемента:

```
private void crossfade() {

        // Изменение прозрачности элемента на 0 и установка значения,
что элемент виден. Необходимо для дальнейшего плавного появления
элемента.
        contentView.setAlpha(0f);
        contentView.setVisibility(View.VISIBLE);

Анимация элемента путем изменения значения свойства alpha,
определяющего прозрачность элемента. Переход к видимому состоянию
элемента 1.
        // listener set on the view.
        contentView.animate()
                .alpha(1f)
                .setDuration(shortAnimationDuration)
                .setListener(null);

        // Анимация элемента интерфейса к значению opacity 0 и
установка значения видимости элемента GONE. ,

        loadingView.animate()
                .alpha(0f)
                .setDuration(shortAnimationDuration)
                .setListener(new AnimatorListenerAdapter() {
        // Обработка завершения анимации. Убираем элемент из верстки.
                    @Override
                    public void onAnimationEnd(Animator animation) {
                        loadingView.setVisibility(View.GONE);
                    }
```

```
                });
    }
```

11.4. Анимация переходов между фрагментами

В классе FragmentTransaction есть несколько готовых анимаций. С помощью метода setTransition(int transit) мы можем указать нужную анимацию и увидеть её в действии. Список готовых анимаций:

- TRANSIT_FRAGMENT_CLOSE;
- TRANSIT_FRAGMENT_OPEN;
- TRANSIT_FRAGMENT_FADE;
- TRANSIT_NONE.

Для того, чтобы установить свою анимацию потребуется использовать метод setCustomAnimations. Методу передаются два параметра. Первый параметр описывает анимацию для фрагмента, который появляется, а второй — описывает анимацию для фрагмента, который убирается с экрана устройства. Метод следует вызывать до появления фрагментов, иначе анимация не будет применена.

Пример использования:

```
fragmentManager.beginTransaction().setCustomAnimations(R.anim.slide_le
ft_in, R,anim.slide_in_right).replace(R.id.container_view,
newFragment).addToBackStack(null).commit();
```

Для того, чтобы показать свою анимацию перехода, необходимо её сначала создать. Делается это с помощью xml файлов, размещаемых в папке res/anim. Примеры своих анимаций.

Сдвиг фрагмента влево:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android" >
    <objectAnimator
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:duration="1500"
        android:interpolator=
            "@android:anim/accelerate_decelerate_interpolator"
        android:propertyName="y"
        android:valueFrom="-1280"
        android:valueTo="0"
        android:valueType="floatType" />
</set>
```

Сдвиг фрагмента вправо:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
     android:ordering="together">

  <objectAnimator
      android:interpolator="@android:anim/accelerate_interpolator"
      android:propertyName="alpha"
      android:valueType="floatType"
      android:valueTo="0"
      android:duration="300"/>

  <objectAnimator
xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator=
        "@android:anim/accelerate_decelerate_interpolator"
    android:propertyName="x"
    android:valueType="floatType"
    android:valueTo="1280"
    android:valueFrom="0"
    android:duration="1500"/>

</set>
```

Элементы визуальных эффектов задаются в теге objectAnimator. У атрибута propertyName указывается свойство фрагмента, которое мы будем изменять при анимации, valueType указывает тип изменяемого параметра. Атрибуты valueFrom и valueTo указывают диапазон изменения параметра, указанного в propertyName. Если параметр valueFrom не указан, то значение берётся равное текущему. В примере значение valueFrom равно -1280, это означает, что движение фрагмента по оси y будет начинаться со значения -1280 и перемещение будет происходить пока значение y не станет равным 0 для верхнего левого угла фрагмента в течение 1500 миллисекунд (атрибут duration).

Тег set служит для объединения эффектов либо их разделения. Он может иметь атрибут android:ordering со значением together, что означает проигрывать эффекты одновременно, в противовес ему существует значение sequentially, которое требует последовательного отображения эффектов в анимации.

11.5. Переворачивание карточки

В некоторых ситуациях необходимо создать элемент, который при нажатии будет переворачиваться и отображать какие-либо данные, например отображение лицевой и оборотной стороны банковской карты.

Для реализации этого потребуется 4 файла анимаций, по две для каждой из сторон карточки. Примеры реализаций (Обратите внимание, что параметры android:duration, как правило выносятся в отдельный файл, хранящий целочисленные значения)

card_flip_left_in.xml:

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Before rotating, immediately set the alpha to 0. -->
    <objectAnimator
        android:valueFrom="1.0"
        android:valueTo="0.0"
        android:propertyName="alpha"
        android:duration="0" />

    <!-- Rotate. -->
    <objectAnimator
        android:valueFrom="-180"
        android:valueTo="0"
        android:propertyName="rotationY"
        android:interpolator="@android:interpolator/accelerate_deceler
ate"
        android:duration="@integer/card_flip_time_full" />

    <!-- Half-way through the rotation (see startOffset), set the
alpha to 1. -->
    <objectAnimator
        android:valueFrom="0.0"
        android:valueTo="1.0"
        android:propertyName="alpha"
        android:startOffset="@integer/card_flip_time_half"
        android:duration="1" />
</set>
```

card_flip_left_out.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Rotate. -->
    <objectAnimator
        android:valueFrom="0"
        android:valueTo="180"
```

```
        android:propertyName="rotationY"
        android:interpolator="@android:interpolator/accelerate_deceler
ate"
        android:duration="@integer/card_flip_time_full" />

    <!-- Half-way through the rotation (see startOffset), set the
alpha to 0. -->
    <objectAnimator
        android:valueFrom="1.0"
        android:valueTo="0.0"
        android:propertyName="alpha"
        android:startOffset="@integer/card_flip_time_half"
        android:duration="1" />
</set>
```

card_flip_right_in.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Before rotating, immediately set the alpha to 0. -->
    <objectAnimator
        android:valueFrom="1.0"
        android:valueTo="0.0"
        android:propertyName="alpha"
        android:duration="0" />

    <!-- Rotate. -->
    <objectAnimator
        android:valueFrom="180"
        android:valueTo="0"
        android:propertyName="rotationY"
        android:interpolator="@android:interpolator/accelerate_deceler
ate"
        android:duration="@integer/card_flip_time_full" />

    <!-- Half-way through the rotation (see startOffset), set the
alpha to 1. -->
    <objectAnimator
        android:valueFrom="0.0"
        android:valueTo="1.0"
        android:propertyName="alpha"
        android:startOffset="@integer/card_flip_time_half"
        android:duration="1" />
</set>
```

card_flip_right_out.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Rotate. -->
    <objectAnimator
        android:valueFrom="0"
        android:valueTo="-180"
        android:propertyName="rotationY"
        android:interpolator="@android:interpolator/accelerate_decelerate"
        android:duration="@integer/card_flip_time_full" />

    <!-- Half-way through the rotation (see startOffset), set the alpha to 0. -->
    <objectAnimator
        android:valueFrom="1.0"
        android:valueTo="0.0"
        android:propertyName="alpha"
        android:startOffset="@integer/card_flip_time_half"
        android:duration="1" />
</set>
```

Потребуется также два макета карточки: для лицевой и оборотной стороны и два класса фрагментов, отвечающих за логику на соответствующей стороне карточки.

Анимация поворота карточки будет задаваться при осуществлении транзакции перехода, как и в предыдущем случае при переходе между фрагментами, с помощью метода setCustomAnimations, однако теперь необходимо передать 4 параметра, по два для каждого из фрагментов.

Пример:

```
setCustomAnimations(
            R.animator.card_flip_right_in,
            R.animator.card_flip_right_out,
            R.animator.card_flip_left_in,
            R.animator.card_flip_left_out)
```

Задание

1. В ранее реализованном приложении для каждого фрагмента создать набор анимация для обработки его появления и исчезновения;
2. Добавить в проект ресурс формата векторной анимации и внедрить его в своё приложение.