Transitions

http://demos.jquerymobile.com/1.0a4/docs/pages/docs-transitions.html

https://www.youtube.com/watch?v=zP_kczi6fHc

https://www.youtube.com/watch?v=K3yMV5am-Xo

Introduction

The Android transitions framework allows you to configure the appearance of changes in your app's user interface. You can animate changes in an app screen, defining each phase as a scene and controlling the way in which the transition changes the app appearance from one scene to another.

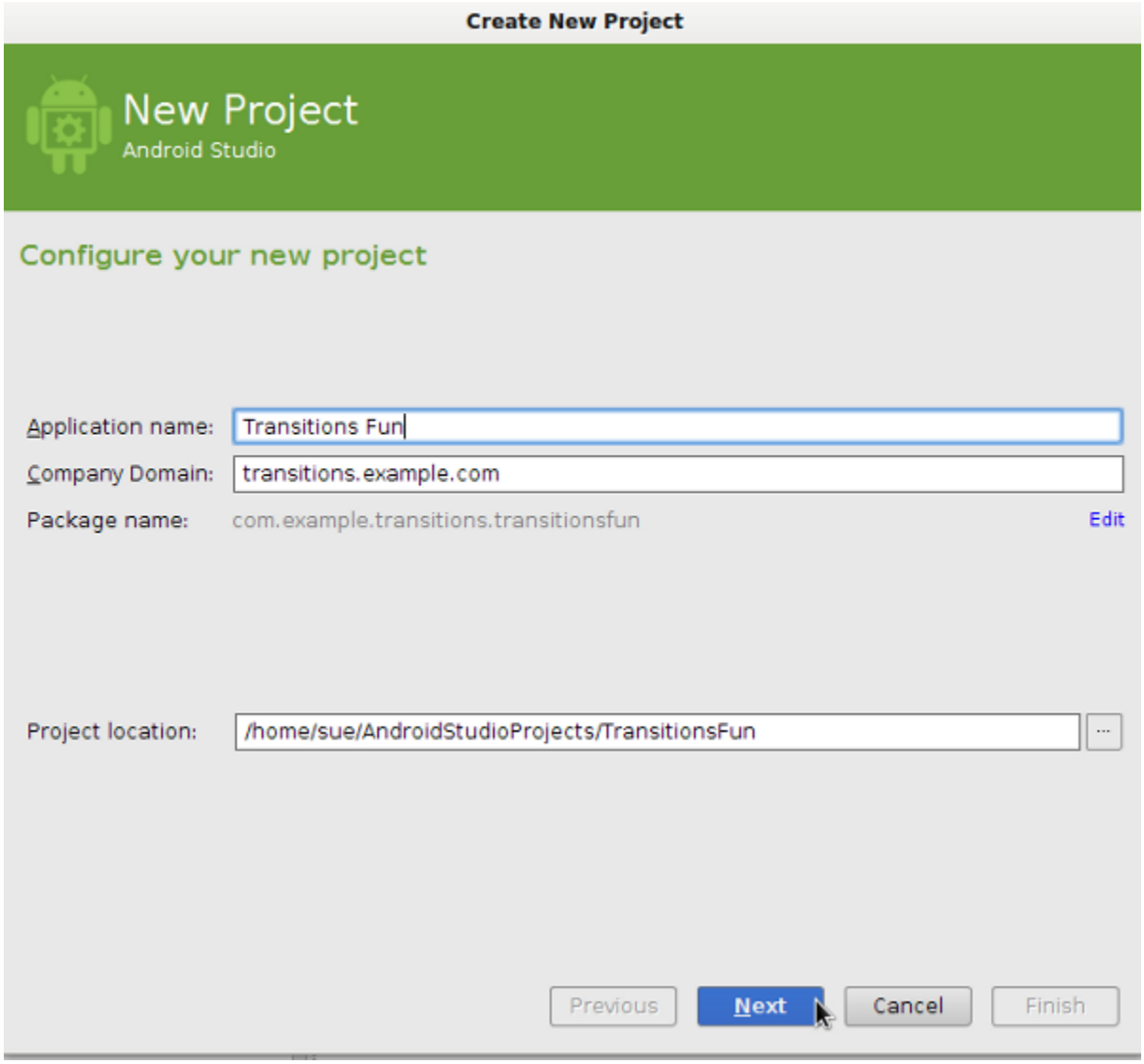
In this tutorial, we will build a simple app with an animated transition in it. This will involve preparing the layout and drawable files in XML, then configuring and applying the transition in Java. We will define two scenes in which the same view items are arranged differently on the screen. As we use a transition, Android will automatically animate the change from one scene to another.

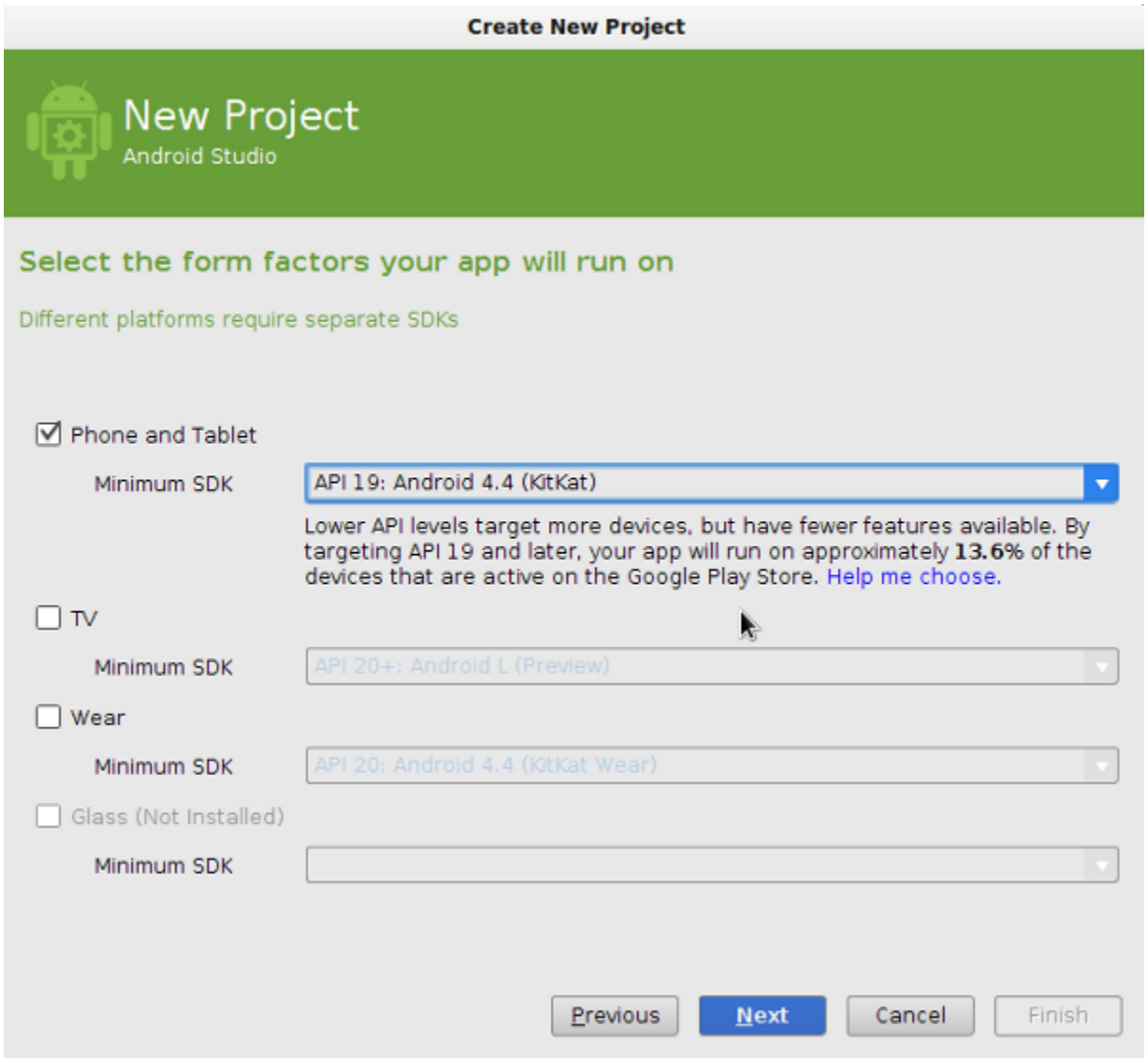
1. Create the App

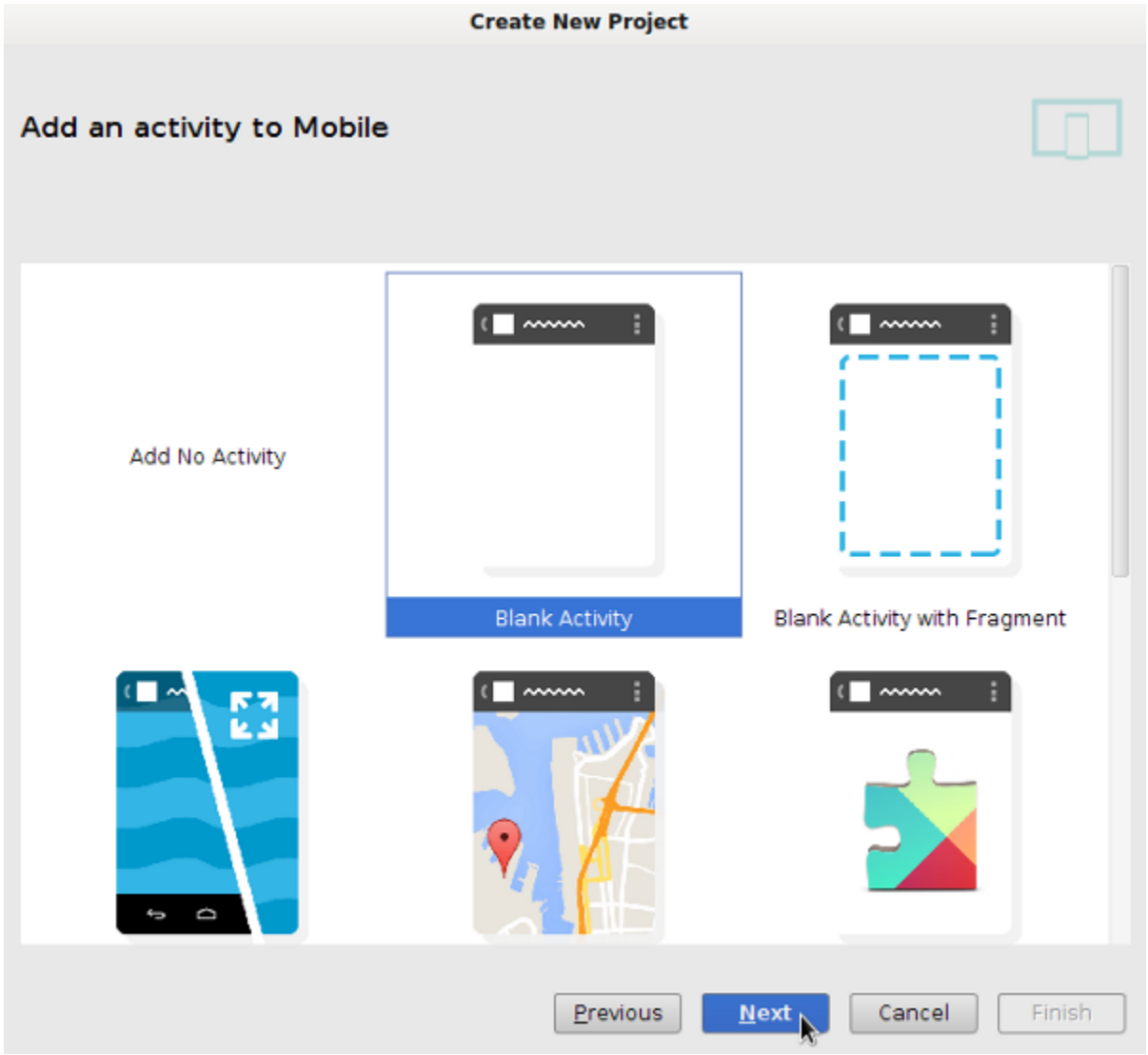
Step 1

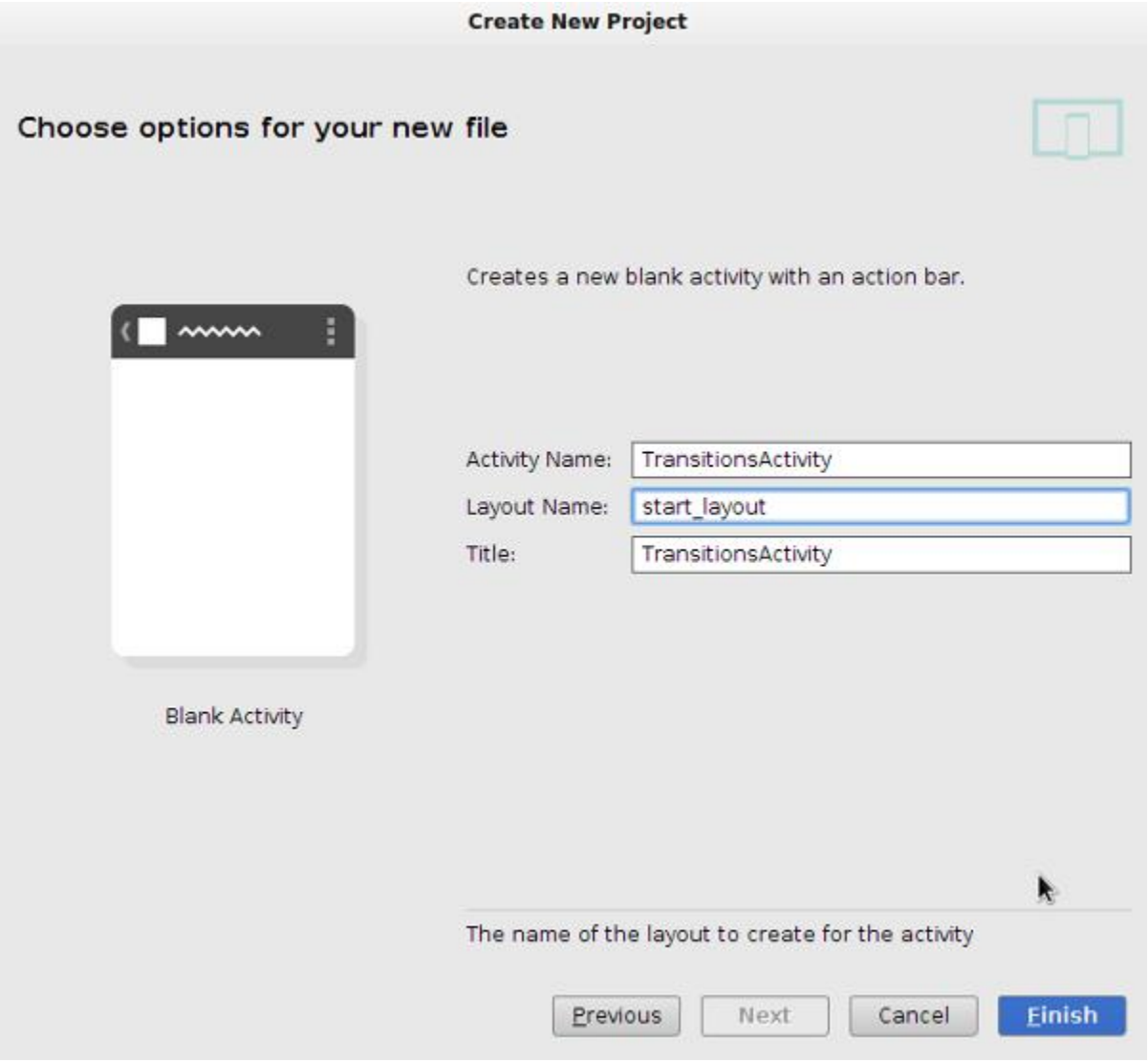
Start by creating a new app in your chosen IDE. You need a minimum SDK of 19 for the transitions classes, so you'll need to take additional steps if you plan on supporting older versions.

Give the app a main `Activity` and layout file, choosing the name **start_layout.xml**for the layout. We will be adding another layout file later, using the transition to change from one to the other. The following images show the process in Android Studio.


Create New Project
New Project
Android Studio
Configure your new project
Application name:
Transitions Fun
Company Domain:
transitions.example.com
Package name:
com.example.transitions.transitionsfun
Edit
Project location:
/home/sue/AndroidStudioProjects/TransitionsFun
Previous
Next
Cancel
Finish


Create New Project
New Project
Android Studio
Select the form factors your app will run on
Different platforms require separate SDKs
Phone and Tablet
Minimum SDK
API 19: Android 4.4 (KitKat)
Lower API levels target more devices, but have fewer features available. By targeting API 19 and later, your app will run on approximately 13.6% of the devices that are active on the Google Play Store. Help me choose.
TV
Minimum SDK
API 20+: Android L (Preview)
Wear
Minimum SDK
API 20: Android 4.4 (KitKat Wear)
Glass (Not Installed)
Minimum SDK
Previous
Next
Cancel
Finish


Create New Project
Add an activity to Mobile
Add No Activity
Blank Activity
Blank Activity with Fragment
Previous
Next
Cancel
Finish



Step 2

Let's now prepare some drawable shapes to use in the transition. We will use four circle shapes with different colored gradient fills. In your app's drawables resource directory, start by creating a new file named **shape1.xml**. Enter the following shape:

```
01 <?xml version="1.0" encoding="utf-8"?>
02 <shape xmlns:android="http://schemas.android.com/apk/res/android"
03     android:dither="true"
04     android:shape="oval" >
05
06     <gradient
           android:endColor="#66ff0000"
```

```
            android:gradientRadius="150"
            android:startColor="#ffffcc00"
            android:type="radial"
            android:useLevel="false" />

        <size
            android:height="100dp"
            android:width="100dp" />

    </shape>
```

The shape is a circle with a radial gradient fill. All four of the shapes will be the same except for the colors used within them. You may wish to create different versions of the drawables for different device densities. Add **shape2.xml** next:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:dither="true"
    android:shape="oval" >

    <gradient
        android:endColor="#66ffcc00"
        android:gradientRadius="150"
        android:startColor="#ff00ff00"
        android:type="radial"
        android:useLevel="false" />

    <size
        android:height="100dp"
```

```
14          android:width="100dp" />
15
16  </shape>
17
```

Now add **shape3.xml**:

```
01
    <?xml version="1.0" encoding="utf-8"?>
02
    <shape xmlns:android="http://schemas.android.com/apk/res/android"
03      android:dither="true"
04      android:shape="oval" >
05
06      <gradient
07          android:endColor="#6600ff00"
08          android:gradientRadius="150"
09          android:startColor="#ff0000ff"
10          android:type="radial"
11          android:useLevel="false" />
12
13      <size
14          android:height="100dp"
15          android:width="100dp" />
16
    </shape>
17
```

Finally add **shape4.xml**:

```
01  <?xml version="1.0" encoding="utf-8"?>
02  <shape xmlns:android="http://schemas.android.com/apk/res/android"
03      android:dither="true"
```

```
        android:shape="oval" >

        <gradient
            android:endColor="#660000ff"
            android:gradientRadius="150"
            android:startColor="#ffff0000"
            android:type="radial"
            android:useLevel="false" />

        <size
            android:height="100dp"
            android:width="100dp" />

    </shape>
```

We will use these shapes as `ImageButtons` in the two layout scenes.

2. Create the Layout Scenes

Step 1

Let's define the two scenes we will transition between as XML layouts. Start with the main layout file you added when you created the app, start_layout.xml. Open it and switch to the XML editing tab. Use a `RelativeLayout` as shown below:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#ff000000"
    android:id="@+id/base"
```

```
    tools:context=".TransitionsActivity">

</RelativeLayout>
```

We add a background color and ID for the layout. The ID is essential to ensure that Android transitions between your scenes, we will be using the same ID in the second scene. When you transition between two scenes, Android will animate the changes as long as each view has the same ID in both scenes, otherwise it will treat the views as different and simply fade them in or out when the transition occurs.

Inside the `RelativeLayout`, add an `ImageButton` for each shape we created:

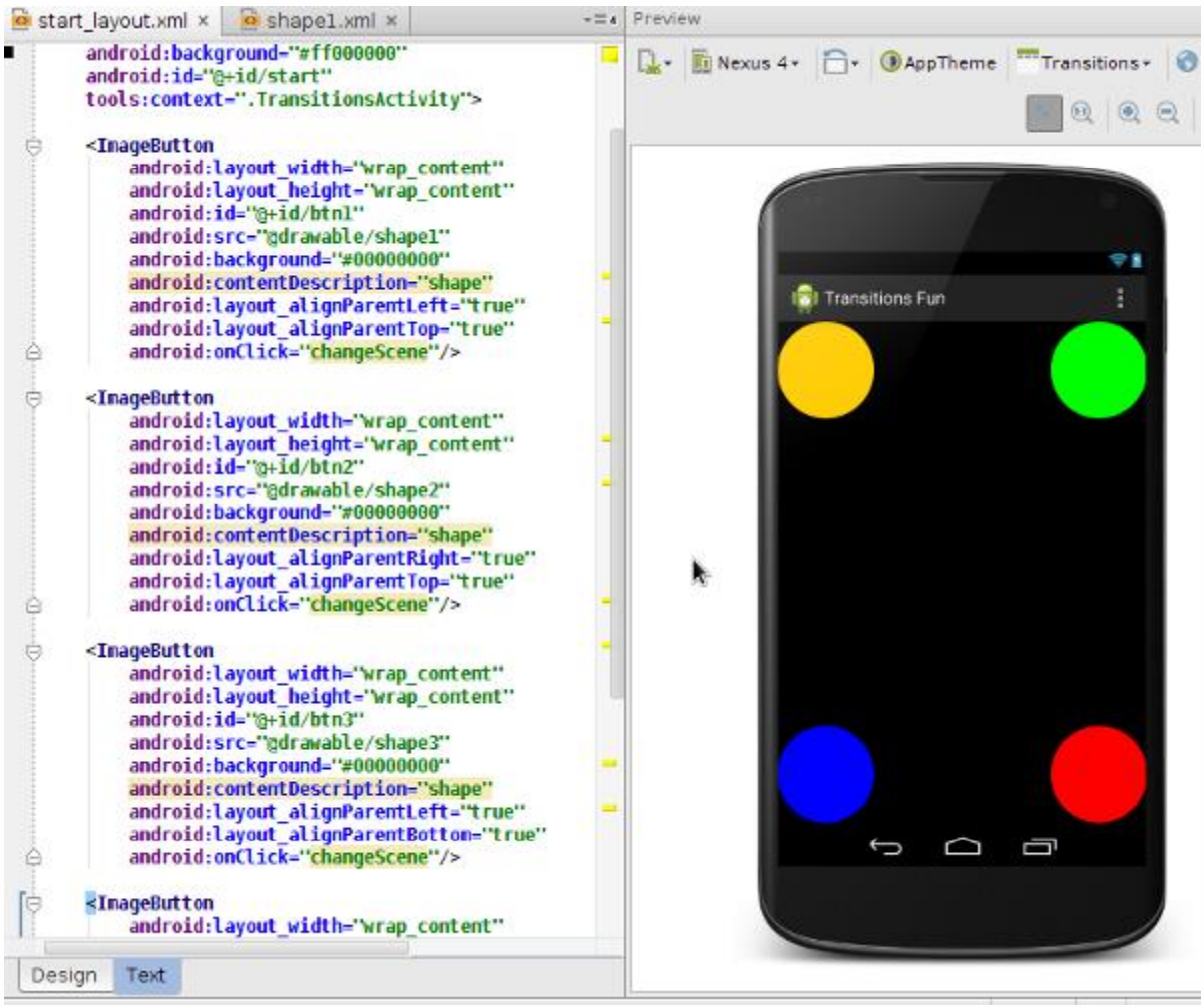
```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/btn1"
    android:src="@drawable/shape1"
    android:background="#00000000"
    android:contentDescription="shape"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:onClick="changeScene"/>

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/btn2"
    android:src="@drawable/shape2"
    android:background="#00000000"
    android:contentDescription="shape"
    android:layout_alignParentRight="true"
```

```
18      android:layout_alignParentTop="true"
19      android:onClick="changeScene"/>
20
21  <ImageButton
22      android:layout_width="wrap_content"
23      android:layout_height="wrap_content"
24      android:id="@+id/btn3"
25      android:src="@drawable/shape3"
26      android:background="#00000000"
27      android:contentDescription="shape"
28      android:layout_alignParentLeft="true"
29      android:layout_alignParentBottom="true"
30      android:onClick="changeScene"/>
31
32  <ImageButton
33      android:layout_width="wrap_content"
34      android:layout_height="wrap_content"
35      android:id="@+id/btn4"
36      android:src="@drawable/shape4"
37      android:background="#00000000"
38      android:contentDescription="shape"
39      android:layout_alignParentRight="true"
40      android:layout_alignParentBottom="true"
41      android:onClick="changeScene"/>
42
43
```

Notice that each shape button has an ID, which will be the same in the second layout we create, and an `onClick` attribute. We will include this method in the main `Activity` later and will start the transition when the user clicks any of the shapes.

You will see a preview of the layout in your IDE, although in some cases the gradient and/or transparency will not be displayed until you actually run the app on a device or the emulator. The shapes are arranged to sit in each corner of the screen as shown below.



Step 2

The first layout we created will represent the start of the transition. Let's now create a second layout file for the scene the transition will change to. Add a new file in your app layout resources directory, naming it **end_layout.xml**. Switch to the text editing tab and enter the following:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#ff000000"
    android:id="@+id/base"
    tools:context=".TransitionsActivity">

    <ImageButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/btn1"
        android:src="@drawable/shape1"
        android:background="#00000000"
        android:contentDescription="shape"
        android:layout_alignParentRight="true"
        android:layout_alignParentBottom="true"
        android:onClick="changeScene"/>

    <ImageButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/btn2"
        android:src="@drawable/shape2"
        android:background="#00000000"
        android:contentDescription="shape"
        android:layout_alignParentLeft="true"
        android:layout_alignParentBottom="true"
        android:onClick="changeScene"/>
```

```
28
29      <ImageButton
30          android:layout_width="wrap_content"
31          android:layout_height="wrap_content"
32          android:id="@+id/btn3"
33          android:src="@drawable/shape3"
34          android:background="#00000000"
35          android:contentDescription="shape"
36          android:layout_alignParentRight="true"
37          android:layout_alignParentTop="true"
38          android:onClick="changeScene"/>
39
40      <ImageButton
41          android:layout_width="wrap_content"
42          android:layout_height="wrap_content"
43          android:id="@+id/btn4"
44          android:src="@drawable/shape4"
45          android:background="#00000000"
46          android:contentDescription="shape"
47          android:layout_alignParentLeft="true"
48          android:layout_alignParentTop="true"
49          android:onClick="changeScene"/>
50
51  </RelativeLayout>
52
53
```

Take a moment to look at the layout code. It is identical to the first layout except for the positions of the shape buttons. Each shape is in the opposite corner from its position in the first layout. The transition will therefore swap the shapes, moving their positions diagonally across the screen.

3. Transition Between Scenes

Step 1

We have the two layouts defined, let's now use a transition to move between them. Open your app's main `Activity` class. You will need the following import statements:

```
import android.transition.AutoTransition;
import android.transition.Scene;
import android.transition.Transition;
import android.view.View;
import android.view.ViewGroup;
import android.view.animation.AccelerateDecelerateInterpolator;
import android.widget.RelativeLayout;
import android.transition.TransitionManager;
```

Inside the `Activity` class declaration, before the `onCreate` method, add the following instance variables we will use to apply the transition:

```
//scenes to transition
private Scene scene1, scene2;
//transition to move between scenes
private Transition transition;
//flag to swap between scenes
private boolean start;
```

Step 2

Now let's prepare for the transition, which will begin when the user clicks a shape. In `onCreate`, after the existing code your IDE has entered, add the following:

```
01  //get the layout ID
02  RelativeLayout baseLayout = (RelativeLayout)findViewById(R.id.base);
03
04  //first scene
05  ViewGroup startViews = (ViewGroup)getLayoutInflater()
06      .inflate(R.layout.start_layout, baseLayout, false);
07
08  //second scene
09  ViewGroup endViews = (ViewGroup)getLayoutInflater()
10      .inflate(R.layout.end_layout, baseLayout, false);
```

First we define the base scene, which is the ID we gave the containing layout in both scene layout files. Next we define the two scenes we are transitioning between, specifying their layout file names and the containing base scene. This will tell Android we want to transition the views within the two scenes, treating any view with the same ID in both scenes as the same object, so that it animates the change from one scene to the other.

Next, we define the two scenes we want to transition between, still in `onCreate`:

```
1  //create the two scenes
2  scene1 = new Scene(baseLayout, startViews);
3  scene2 = new Scene(baseLayout, endViews);
```

We pass the base layout and relevant scene layouts to each constructor. Now we can refer to these scenes when defining the transition.

Step 3

Let's get the transition prepared, still in `onCreate`:

```
1  //create transition, set properties
2  transition = new AutoTransition();
3  transition.setDuration(5000);
4  transition.setInterpolator(new AccelerateDecelerateInterpolator());
5
6  //initialize flag
7  start=true;
```

Android provides a range of transition types depending on how you want the changes in your scenes to be animated. In this case, we choose an `AutoTransition`, so Android will calculate how to make the change based on the properties that are altered between scenes. See the Transitions reference for more options.

We set a duration and interpolator for the transition. You can optionally also set a start delay for the change. Finally, we initialize the boolean flag to true. For simplicity we will use this to swap between the two scenes whenever the user clicks a shape, but this is just to demonstrate the functionality involved.

Step 4

Remember that we added an `onClick` attribute to the shape buttons when we created the layout XML. Let's add that method to the `Activity` now:

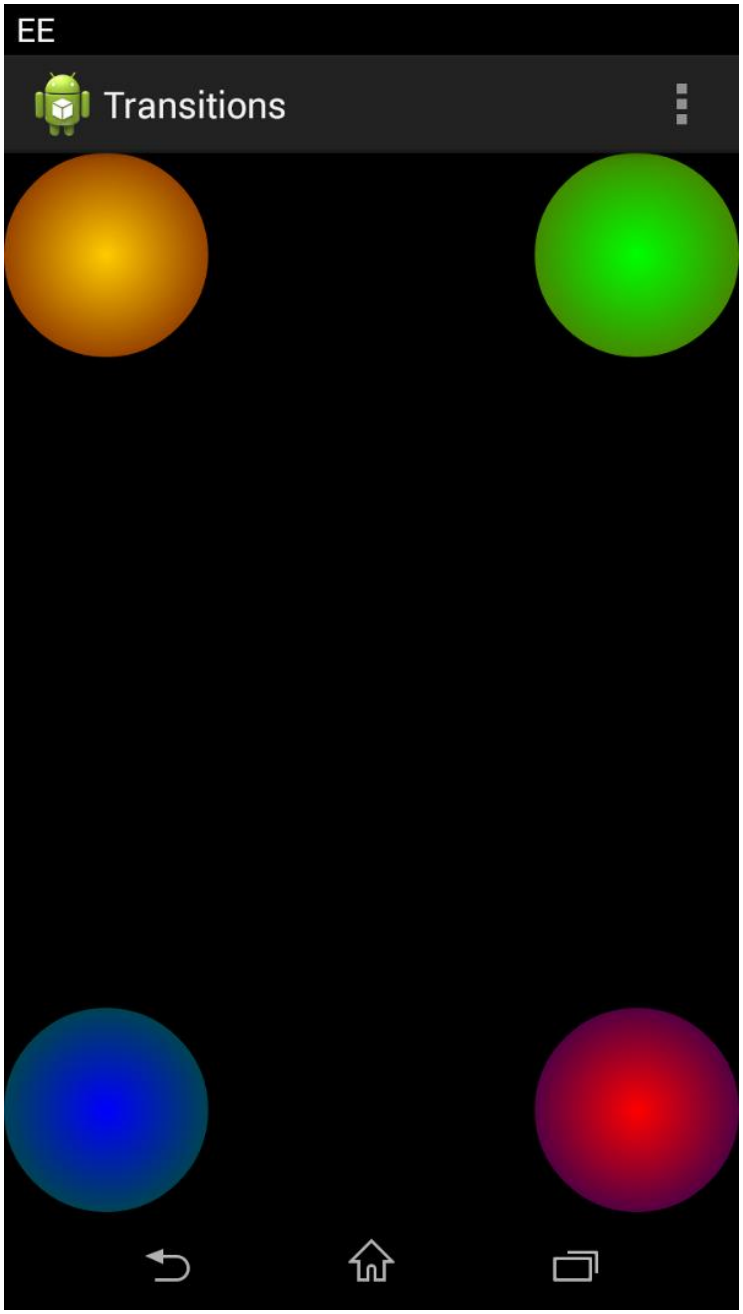
```
01  public void changeScene(View v){
02
03      //check flag
04      if(start) {
05          TransitionManager.go(scene2, transition);
06          start=false;
07      }
      else {
```

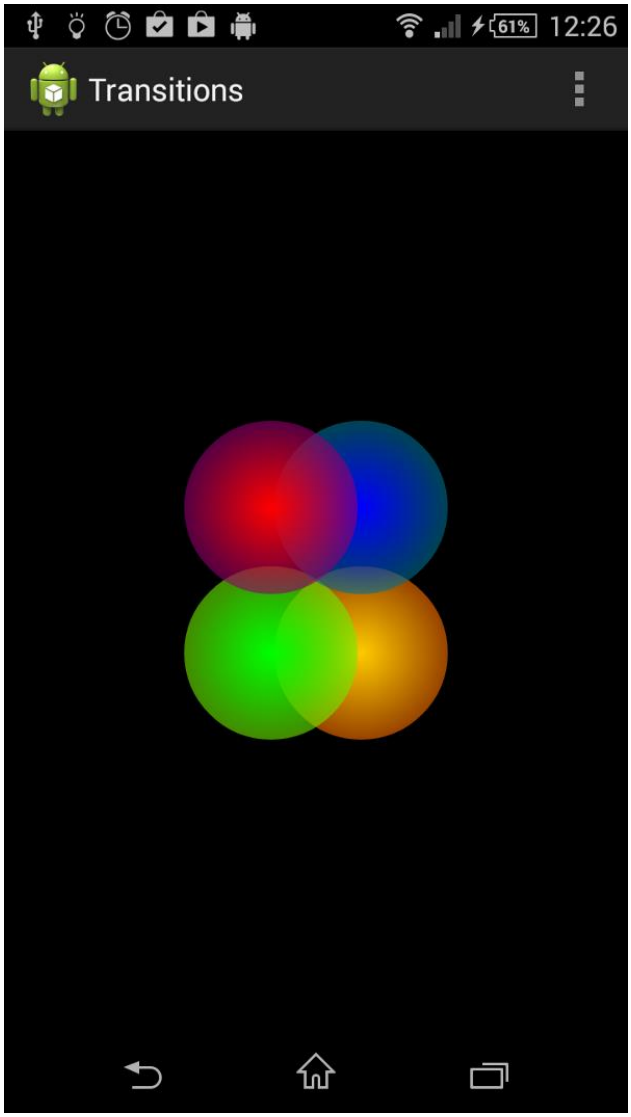
```
            TransitionManager.go(scene1, transition);
            start=true;
        }
    }

```

We use the `TransitionManager` to transition from the current scene to the other scene, with the boolean flag keeping track of which one we are on. We specify the `Transition` object we created to tailor how the change unfolds.

You should now be able to run your app and see the transition whenever you click a shape. Each time you click, the transition should move the shapes slowly to the opposite corners, then swap them back when you click again.


EE
Transitions



Conclusion

In this tutorial we have really only begun to explore what you can do with the Android transitions framework. To develop your transitions further, check out the additional methods in the `TransitionManager` class, such as `beginDelayedTransition` and `transitionTo`. You can also use a `TransitionSet` to combine multiple transitions, for example, to configure fading and moving effects. Depending on the complexity of your transitions, you may also benefit from the `TransitionValues` class, which provides a reference to data values relevant to the transition. For more on what you can do with scenes, check out the `Scene` class as well.