Frame animation and Tween animation in android

Objectives:

- To provide a good understanding of how to use Frame animation and Tween animation in android on any View.
- To provide knowledge of some attributes of translate, rotate and scale tags.
- To provide familiarity with Animation Class.

Animations add vivacity and personality to your apps. There are two types of animations that you can do with the Animation framework of Android: Frame animation and Tween animation. Frame animation is series of frames is drawn one after the other at regular. Tween animation is simple transformations of position, size, rotation to the content of a View. Animation can be defined in XML that performs transitions such as rotating, fading, moving, and stretching on a graphic.

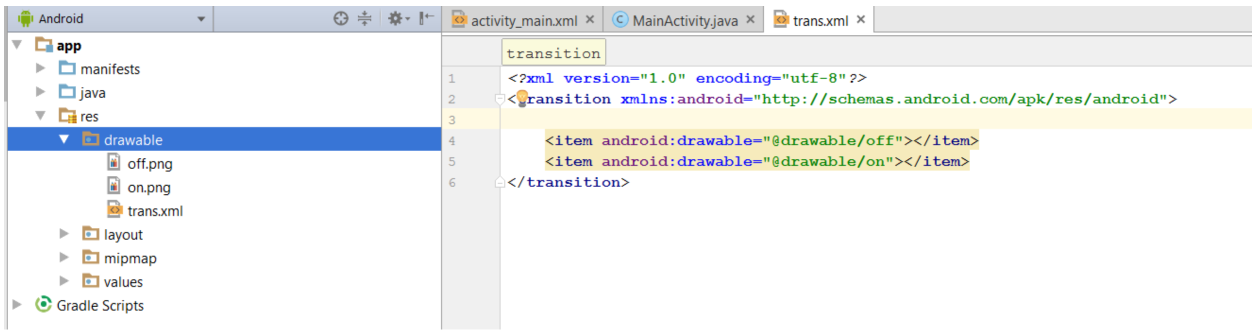
Procedure:

1-Frame Animation:

The idea behind a frame animation is simple: We'll be cycling through a series of images very quickly, just like an old movie reel. The "frame" refers to a single image. Thus, the first step in creating a custom frame animation is to create a sequence of images. We have two options here: we can use XML drawable (such as shape drawable) or actual image files. For the sake of simplicity, we're going to use the following series of PNG images:

1-You should make sure to have images sized appropriately for different screen densities. For now, shove those images into the res/drawable folder.

2-Now that we have our images to cycle through, the next step is to define an XML Drawable for our animation. We can use transition to make the Frame Animation. It just cycles through a sequence of provided images. Fig1 shows an example of an Frame Animation for changing the state of a lamp. Each item in the list is just pointing to one of the images in our sequence from earlier. All we have to do is place them in the correct order.



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

    <item android:drawable="@drawable/off"></item>
    <item android:drawable="@drawable/on"></item>
</transition>
```

Fig1: Using transition to make frame animation.

3-In the layout of main activity, use the Linear Layout as the root element and put a Button and an ImageView inside it. You should use the trans.xml as the source of the ImageView component.

```
<?xml version="1.0" encoding="utf-8"?>
<android.widget.LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.example.mohammad_ateeq.animation.MainActivity">

        <ImageView
            android:id="@+id/imageView2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:srcCompat="@drawable/trans"
            android:layout_weight="0.74" />

        <Button
            android:id="@+id/button"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="0.27"
            android:text="GO" />

</android.widget.LinearLayout>
```

4- In our Activity, we grab a reference to the ImageView and then start the animation, like so:

```
package com.example.mohammad_ateeq.animation;
import android.graphics.drawable.TransitionDrawable;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final ImageView imageView = (ImageView)
findViewById(R.id.imageView2);
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
               TransitionDrawable transitionDrawable= (TransitionDrawable)
imageView.getDrawable();
                transitionDrawable.startTransition(3000);
            }
        });
    }
}
```

5-

When you run the application, you should see the transition between the two images like fig2.

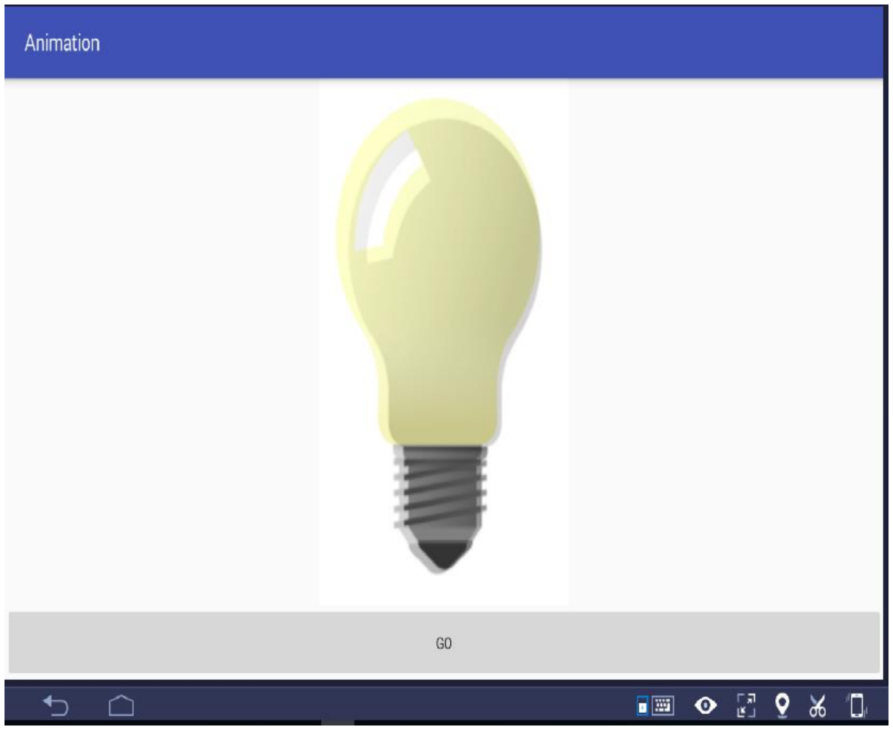


Fig2: The transition between two images.

2- Tween Animation.

Tween Animation is defined as an animation which is used to Translate, Rotate, Scale and Alpha any type of view in Android.

Step 1: Create a new folder in res directory and name it anim.

Step 2: Create three XML files in anim folder and copy an image to drawable folder as shown in figure3.

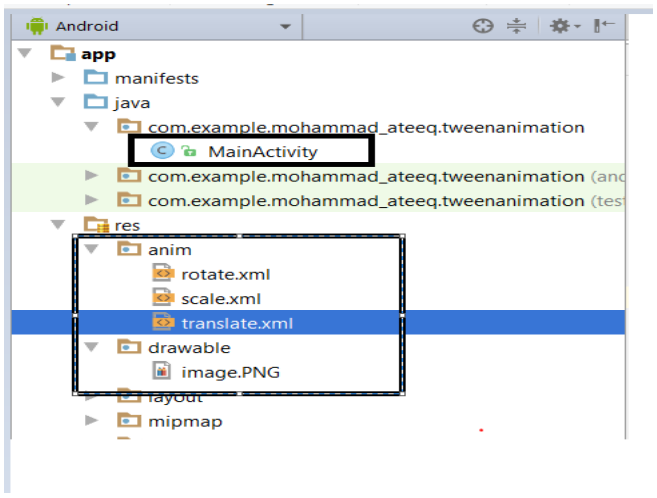

Android
app
manifests
java
com.example.mohammad_ateeq.tweenanimation
MainActivity
com.example.mohammad_ateeq.tweenanimation (and
com.example.mohammad_ateeq.tweenanimation (test
res
anim
rotate.xml
scale.xml
translate.xml
drawable
image.PNG
layout
mipmap

Figure3: anim and drawable folders

Step 3: Paste the following code in your main.xml layout file. We use a Linear layout as parent layout which includes another two layouts: The first layout contains three buttons and the second layout contains ImageView component.

```xml
<?xml version="1.0" encoding="utf-8"?>
<android.widget.LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:orientation="vertical"
    android:layout_height="match_parent"
    tools:context="com.example.mohammad_ateeq.tweenanimation.MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
            android:id="@+id/rotatebutton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Rotate" />

        <Button
            android:id="@+id/zoombutton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Zoom" />

        <Button
            android:id="@+id/translatebutton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Translate" />
    </LinearLayout>

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="fill_parent"
       >

        <ImageView
            android:id="@+id/imageView2"
            android:layout_width="85dp"
            android:layout_height="90dp"
            app:srcCompat="@drawable/image"
            android:layout_centerHorizontal="true"
            android:layout_centerVertical="true"/>
    </RelativeLayout>
</android.widget.LinearLayout>
```

Step 4: Now copy the following code in your rotate.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
<rotate
    android:fromDegrees="0"
    android:toDegrees="360"
    android:pivotY="50%"
    android:pivotX="50%"
    android:duration="3000">

</rotate>
</set>
```

The above code defines a rotate animation which is designed by using its various attributes.

- xmlns:android: This attribute must be defined in the root tag to get the schema path of this XML file.
- You can specify the center point of the rotation by specifying pivotX and pivotY.

- android:duration: This attribute is used to define the duration in which the animation needs to be completed in milliseconds.

Step 5: Copy the following code in your translate.xml file

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

    <translate
    android:fromXDelta="0%p"
    android:toXDelta="30%p"
    android:duration="3000"
    />
</set>
```

The above code defines a translate animation which is designed by using its various attributes.

- android:fromXDelta: This attribute is used to define the starting point of translation animation on Xaxis.
- android:toXDelta: This attribute is used to define the ending point of translation animation on Xaxis.
- Values from -100 to 100 ending with "%", indicating a percentage relative to itself.
- Values from -100 to 100 ending in "%p", indicating a percentage relative to its parent.
- A float value with no suffix, indicating an absolute value.

Step 6: Copy the following code in your scale.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
<scale
    android:fromXScale="1"
    android:fromYScale="1"
    android:toXScale="3"
    android:toYScale="3"
    android:pivotX="50%"
    android:pivotY="50%"
    android:duration="2000"
    ></scale>
</set>
```

The above code defines a scale animation which is designed by using its various attributes.

- android:fromXScale: Starting X size offset, where 1.0 is no change.
- android:toXScale: Ending X size offset, where 1.0 is no change.
- android:fromYScale: Starting Y size offset, where 1.0 is no change.
- android:toYScale: Ending Y size offset, where 1.0 is no change.
- android:pivotX: The X coordinate to remain fixed when the object is scaled.
- android:pivotY:The Y coordinate to remain fixed when the object is scaled.

Step7: Paste the following code in your MainActivity.java file :

```
package com.example.mohammad_ateeq.tweenanimation;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.Button;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button translateButton= (Button) findViewById(R.id.translatebutton);
        final ImageView imageView = (ImageView) findViewById(R.id.imageView2);
        translateButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Animation animation =
AnimationUtils.loadAnimation(MainActivity.this,R.anim.translate);
                imageView.startAnimation(animation);

            }
        });
        Button zoomButton= (Button) findViewById(R.id.zoombutton);
        zoomButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Animation animation =
AnimationUtils.loadAnimation(MainActivity.this,R.anim.scale);
                imageView.startAnimation(animation);
            }
        });

        Button rotateButton= (Button) findViewById(R.id.rotatebutton);
        rotateButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Animation animation =
AnimationUtils.loadAnimation(MainActivity.this,R.anim.rotate);
                imageView.startAnimation(animation);
            }
        });


    }
}
```

- Animation Class : Animation class is used to hold an animation which is loaded in it from the anim folder of resource directory. In the above code, the animation reference variable holds the animation xml files loaded by the use of AnimationUtils class.

- AnimationUtils : AnimationUtils class is used to fetch an animation file from anim folder by using loadAnimation method of this class which has two arguments: First argument defines context in which the animation is to be loaded which must be the context of given activity on which the animation is to be loaded. Second argument defines the id of the animation in resources file.

Step 8: Now, you can run the application. The output should look like fig4.

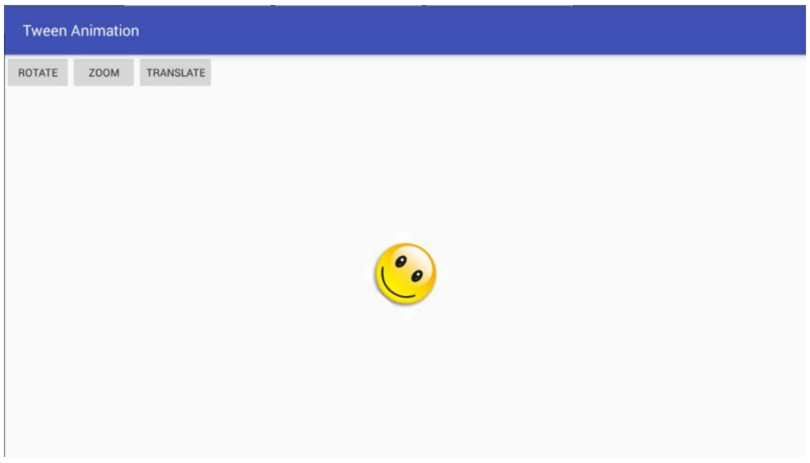


Fig4: the output of the application.

TODO1: Add “Fade In” animation to the ImageView. For fade in animation you can use <alpha> tag which defines alpha value. Create new xml file (fade_in.xml) and paste the following code inside it.

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

    <alpha
        android:duration="1000"
        android:fromAlpha="0.0"
        android:toAlpha="1.0" />

</set>
```

TODO2: Add Sequential Animation to the application

If you want to perform multiple animation in a sequential manner you have to use android:startOffset to give start delay time. The easy way to calculate this value is to add the duration and startOffset values of previous animation. Following is a sequential animation where set of move animations performs in sequential manner.

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:fillAfter="true"
    android:interpolator="@android:anim/linear_interpolator" >

    <!-- Use startOffset to give delay between animations -->


    <!-- Move -->
    <translate
        android:duration="800"
        android:fillAfter="true"
        android:fromXDelta="0%p"
        android:startOffset="300"
        android:toXDelta="75%p" />
    <translate
        android:duration="800"
        android:fillAfter="true"
        android:fromYDelta="0%p"
        android:startOffset="1100"
        android:toYDelta="70%p" />
    <translate
        android:duration="800"
        android:fillAfter="true"
        android:fromXDelta="0%p"
        android:startOffset="1900"
        android:toXDelta="-75%p" />
    <translate
        android:duration="800"
        android:fillAfter="true"
        android:fromYDelta="0%p"
        android:startOffset="2700"
        android:toYDelta="-70%p" />

    <!-- Rotate 360 degrees -->
    <rotate
        android:duration="1000"
        android:fromDegrees="0"
        android:interpolator="@android:anim/cycle_interpolator"
        android:pivotX="50%"
        android:pivotY="50%"
        android:startOffset="3800"
        android:repeatCount="infinite"
        android:repeatMode="restart"
        android:toDegrees="360" />

</set>
```