# 1. Android Animations

## 1.1. Using animations

Android allows changing object properties over a certain time interval via the properties animation API.

The superclass of the animation API is the `Animator` class. The `ObjectAnimator` class can be used to modify attributes of an object.

You can also add an `AnimatorListener` class to your `Animator` class. This listener is called in the different phases of the animation. You can use this listener to perform actions before or after a certain animation, e.g. add or remove a `View` from a `ViewGroup`.

The `animate()` method on a `View` object returns an `ViewPropertyAnimator` object for the view. It provides a fluent API to typical animations which can be performed on views.

The following code shows an example.

```
JAVA
myView.animate().translationX(400);

// if an animation is slow you can try to activate a hardware layer which
// uses a cache
// watch-out: this might not always result in a correct animation

myView.animate().translationX(400).withLayer();
```

You can also register action, which are execute fore the start or after the end of the animation.

```
JAVA
// StartAction
myView.animate().translationX(100).withStartAction(new Runnable(){
    public void run(){
        viewer.setTranslationX(100-myView.getWidth());
        // do something
    }
});

// EndAction
myView.animate().alpha(0).withEndAction(new Runnable(){
    public void run(){
        // rRemove the view from the parent layout
        parent.removeView(myView);
    }
});
```

## 1.2. Define the rate of change for an animation

Via the `setInterpolator()` method you register an `TimeInterpolator` object with an animation. It defines the rate of change for an animation.

The standard is linear. The Android platform defines a few default ones. For example, the `AccelerateDecelerateInterpolator` class defines that the animation starts and ends slowly but accelerates through the middle.

## 1.3. Using animations with arbitrary properties

The animation system cannot automatically understand every type. Via the `setEvaluator` method you can set an object of type `TypeEvaluator`. It allows creating animations for arbitrary types, by providing custom evaluators for these

types.

### 1.4. Layout animations

The `LayoutTransition` class allows setting animations on a layout container and a change on the view hierarchy of this container will be animated.

JAVA

```
package com.example.android.layoutanimation;

import android.animation.LayoutTransition;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;

public class MainActivity extends Activity {

    private ViewGroup viewGroup;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        LayoutTransition l = new LayoutTransition();
        l.enableTransitionType(LayoutTransition.CHANGING);
        viewGroup = (ViewGroup) findViewById(R.id.container);
        viewGroup.setLayoutTransition(l);

    }

    public void onClick(View view) {
        viewGroup.addView(new Button(this));
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }
}
```

### 1.5. Animations for Activity transition

Animations can be applied to `Views` but it is also possible to apply them on the transition between activities.

The `ActivityOptions` class allows defining defaults or customer animations.

JAVA

```
public void onClick(View view) {
    Intent intent = new Intent(this, SecondActivity.class);
    ActivityOptions options = ActivityOptions.makeScaleUpAnimation(view, 0,
            0, view.getWidth(), view.getHeight());
    startActivity(intent, options.toBundle());
}
```

## 2. Android Basics

The following description assumes that you have already basic knowledge in Android development.

Please check https://www.vogella.com/tutorials/Android/article.html - Android development tutorial, to learn the basics. Also see https://www.vogella.com /android.html - Android development tutorials, for more information about Android development.

## 3. Exercise: Using the properties animations API

This exercise demonstrates the usage of the Properties animation API.

Create a new Android project with the top level package name *com.vogella.android.animation.views* and an *activity* called *AnimationExampleActivity*. The layout file should be called *activity_main.xml*.

Change your layout file to the following code.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <LinearLayout
        android:id="@+id/test"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" >

        <Button
            android:id="@+id/Button01"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="startAnimation"
            android:text="Rotate" />

        <Button
            android:id="@+id/Button04"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="startAnimation"
            android:text="Group" >
        </Button>

        <Button
            android:id="@+id/Button03"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="startAnimation"
            android:text="Fade" />

        <Button
            android:id="@+id/Button02"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="startAnimation"
            android:text="Animate" />

    </LinearLayout>

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:src="@drawable/icon" />

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/imageView1"
        android:layout_alignRight="@+id/imageView1"
        android:layout_marginBottom="30dp"
        android:text="Large Text"
        android:textAppearance="?android:attr/textAppearanceLarge" />

</RelativeLayout>
```

Create the following menu resource.

XML

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

    <item
        android:id="@+id/item1"
        android:showAsAction="ifRoom"
        android:title="Game">
    </item>

</menu>
```

Change your activity similar to the following listing.

JAVA

```
package com.vogella.android.animation.views;

import android.animation.AnimatorSet;
import android.animation.ObjectAnimator;
import android.app.Activity;
import android.content.Intent;
import android.graphics.Paint;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ImageView;
import android.widget.TextView;

public class AnimationExampleActivity extends Activity {

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

    }

    public void startAnimation(View view) {
        float dest = 0;
        ImageView aniView = (ImageView) findViewById(R.id.imageView1);
        switch (view.getId()) {

        case R.id.Button01:
            dest = 360;
            if (aniView.getRotation() == 360) {
                System.out.println(aniView.getAlpha());
                dest = 0;
            }
            ObjectAnimator animation1 = ObjectAnimator.ofFloat(aniView,
                    "rotation", dest);
            animation1.setDuration(2000);
            animation1.start();
            // Show how to load an animation from XML
            // Animation animation1 = AnimationUtils.loadAnimation(this,
            // R.anim.myanimation);
            // animation1.setAnimationListener(this);
            // animatedView1.startAnimation(animation1);
            break;

        case R.id.Button02:
            // shows how to define a animation via code
            // also use an Interpolator (BounceInterpolator)
            Paint paint = new Paint();
            TextView aniTextView = (TextView) findViewById(R.id.textView1);
            float measureTextCenter = paint.measureText(aniTextView.getText()
                    .toString());
            dest = 0 - measureTextCenter;
            if (aniTextView.getX() < 0) {
                dest = 0;
            }
            ObjectAnimator animation2 = ObjectAnimator.ofFloat(aniTextView,
                    "x", dest);
            animation2.setDuration(2000);
            animation2.start();
            break;

        case R.id.Button03:
            // demonstrate fading and adding an AnimationListener

            dest = 1;
            if (aniView.getAlpha() > 0) {
                dest = 0;
            }
            ObjectAnimator animation3 = ObjectAnimator.ofFloat(aniView,
                    "alpha", dest);
            animation3.setDuration(2000);
            animation3.start();
```

```
            break;

        case R.id.Button04:

            ObjectAnimator fadeOut = ObjectAnimator.ofFloat(aniView, "alpha",
                    0f);
            fadeOut.setDuration(2000);
            ObjectAnimator mover = ObjectAnimator.ofFloat(aniView,
                    "translationX", -500f, 0f);
            mover.setDuration(2000);
            ObjectAnimator fadeIn = ObjectAnimator.ofFloat(aniView, "alpha",
                    0f, 1f);
            fadeIn.setDuration(2000);
            AnimatorSet animatorSet = new AnimatorSet();

            animatorSet.play(mover).with(fadeIn).after(fadeOut);
            animatorSet.start();
            break;

        default:
            break;
        }

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.mymenu, menu);
        return super.onCreateOptionsMenu(menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        Intent intent = new Intent(this, HitActivity.class);
        startActivity(intent);
        return true;
    }
}
```

Create a new xml layout file called *target* like the following.

JAVA

```
<?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">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button1"/>

</LinearLayout>
```

Create a new `HitActivity` activity.

JAVA

```
package com.vogella.android.animation.views;

import java.util.Random;

import android.animation.Animator;
import android.animation.AnimatorListenerAdapter;
import android.animation.AnimatorSet;
import android.animation.ObjectAnimator;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class HitActivity extends Activity {
    private ObjectAnimator animation1;
    private ObjectAnimator animation2;
    private Button button;
    private Random randon;
    private int width;
    private int height;
    private AnimatorSet set;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.target);
        width = getWindowManager().getDefaultDisplay().getWidth();
        height = getWindowManager().getDefaultDisplay().getHeight();
        randon = new Random();

        set = createAnimation();
        set.start();
        set.addListener(new AnimatorListenerAdapter() {

            @Override
            public void onAnimationEnd(Animator animation) {
                int nextX = randon.nextInt(width);
                int nextY = randon.nextInt(height);
                animation1 = ObjectAnimator.ofFloat(button, "x",
button.getX(),
                        nextX);
                animation1.setDuration(1400);
                animation2 = ObjectAnimator.ofFloat(button, "y",
button.getY(),
                        nextY);
                animation2.setDuration(1400);
                set.playTogether(animation1, animation2);
                set.start();
            }
        });
    }

    public void onClick(View view) {
        String string = button.getText().toString();
        int hitTarget = Integer.valueOf(string) + 1;
        button.setText(String.valueOf(hitTarget));
    }

    private AnimatorSet createAnimation() {
        int nextX = randon.nextInt(width);
        int nextY = randon.nextInt(height);
        button = (Button) findViewById(R.id.button1);
        animation1 = ObjectAnimator.ofFloat(button, "x", nextX);
        animation1.setDuration(1400);
        animation2 = ObjectAnimator.ofFloat(button, "y", nextY);
        animation2.setDuration(1400);
        AnimatorSet set = new AnimatorSet();
        set.playTogether(animation1, animation2);
        return set;
    }
}
```

If you run this example and press the different buttons, the animation should start.

Via the toolbar, you can start your `HitActivity`.

## 4. Activity animations in Android with shared views

Android 5.0 adds the capability to animate between activities and to have shared views between these activity. If you define a shared part the old view with be animating into the position and size of the new view.

To test this create a project with the top level package called `com.vogella.android.activityanimationwithsharedviews`.

Create two activity with two different layout, both containing a `ImageView` with the same `android:transitionName` property.

activity_main.xml

XML

```
<LinearLayout 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"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/sharedimage"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:scaleType="centerCrop"
        android:src="@drawable/ic_sharedimage"
        />

</LinearLayout>
```

activity_second.xml

XML

```
<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"

tools:context="com.vogella.android.activityanimationwithsharedviews.SecondAc
tivity">

    <ImageView
        android:id="@+id/sharedimage"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_sharedimage"
        android:layout_alignParentBottom="true"
        android:layout_alignParentEnd="true" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello_world"
        android:id="@+id/textView" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Button"
        android:id="@+id/button"
        android:transitionName="sharedImage"
        android:layout_below="@+id/textView"
        android:layout_alignParentStart="true"
        android:layout_marginTop="54dp" />

</RelativeLayout>
```

Adjust your activity code.

JAVA

```
package com.vogella.android.activityanimationwithsharedviews;

import android.app.Activity;
import android.app.ActivityOptions;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageView;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final ImageView sharedImage = (ImageView)
findViewById(R.id.sharedimage);
        sharedImage.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                //This is where the magic happens.
                // makeSceneTransitionAnimation takes a context, view,
                // a name for the target view.
                ActivityOptions options =
                        ActivityOptions.
                        makeSceneTransitionAnimation(MainActivity.this,
sharedImage, "sharedImage");
                Intent intent = new Intent(MainActivity.this,
SecondActivity.class);
                startActivity(intent, options.toBundle());
            }
        });

    }

}
```

JAVA

```
package com.vogella.android.activityanimationwithsharedviews;

import android.app.Activity;
import android.os.Bundle;

public class SecondActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
    }
}
```

If you run your application and click on the image view, it is animated to the view with the same `android:transitionName` property, in our case the button.