Animation APIs

The API allows to define for arbitrary object properties a start and end value and apply a time-based change to this attribute. This API can be applied on any Java object not only on Views.

Animator and AnimatorListener

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

The ViewPropertyAnimator class introduced in Android 3.1 provides a simpler access to typical animations which are performed on Views.

The animate() method on a View will return the ViewPropertyAnimator object. This object allows to perform simultaneous animations. It has a fluent API and allows to set the duration of the animation.

The target of ViewPropertyAnimator is to provide a very simple API for typical animations.

he following code shows an example of the usage of this method.

```
// Using hardware layer
myView.animate().translationX(400).withLayer();
```

For performance optimization you can also let ViewPropertyAnimator use a hardware layout.

```
// Using hardware layer
myView.animate().translationX(400).withLayer();
```

You can also directly define a Runnable to be executed at the start and the end of the animation.

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

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

1.4. Layout animations

The LayoutTransition class allows to set animations on a layout container and a changes on the Views of this container will be animated.

```
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;
  }
}
```

Animations for Activity transition **PAGE TOP**

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

The ActivityOptions class allows to define default or customer animations.

```
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());
}
```