/\*

\* Copyright 2012 The Android Open Source Project

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package com.example.android.animationsdemo;

import android.animation.Animator;

import android.animation.AnimatorListenerAdapter;

import android.animation.AnimatorSet;

import android.animation.ObjectAnimator;

import android.content.Intent;

import android.graphics.Point;

import android.graphics.Rect;

import android.os.Bundle;

import android.support.v4.app.FragmentActivity;

import android.support.v4.app.NavUtils;

import android.view.MenuItem;

import android.view.View;

import android.view.animation.DecelerateInterpolator;

import android.widget.ImageView;

/\*\*

\* A sample showing how to zoom an image thumbnail to full-screen, by animating the bounds of the

\* zoomed image from the thumbnail bounds to the screen bounds.

\*

\* <p>In this sample, the user can touch one of two images. Touching an image zooms it in, covering

\* the entire activity content area. Touching the zoomed-in image hides it.</p>

\*/

public class ZoomActivity extends FragmentActivity {

/\*\*

\* Hold a reference to the current animator, so that it can be canceled mid-way.

\*/

private Animator mCurrentAnimator;

/\*\*

\* The system "short" animation time duration, in milliseconds. This duration is ideal for

\* subtle animations or animations that occur very frequently.

\*/

private int mShortAnimationDuration;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_zoom);

// Hook up clicks on the thumbnail views.

final View thumb1View = findViewById(R.id.thumb\_button\_1);

thumb1View.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

zoomImageFromThumb(thumb1View, R.drawable.image1);

}

});

final View thumb2View = findViewById(R.id.thumb\_button\_2);

thumb2View.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

zoomImageFromThumb(thumb2View, R.drawable.image2);

}

});

// Retrieve and cache the system's default "short" animation time.

mShortAnimationDuration = getResources().getInteger(android.R.integer.config\_shortAnimTime);

}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

switch (item.getItemId()) {

case android.R.id.home:

// Navigate "up" the demo structure to the launchpad activity.

// See http://developer.android.com/design/patterns/navigation.html for more.

NavUtils.navigateUpTo(this, new Intent(this, MainActivity.class));

return true;

}

return super.onOptionsItemSelected(item);

}

/\*\*

\* "Zooms" in a thumbnail view by assigning the high resolution image to a hidden "zoomed-in"

\* image view and animating its bounds to fit the entire activity content area. More

\* specifically:

\*

\* <ol>

\* <li>Assign the high-res image to the hidden "zoomed-in" (expanded) image view.</li>

\* <li>Calculate the starting and ending bounds for the expanded view.</li>

\* <li>Animate each of four positioning/sizing properties (X, Y, SCALE\_X, SCALE\_Y)

\* simultaneously, from the starting bounds to the ending bounds.</li>

\* <li>Zoom back out by running the reverse animation on click.</li>

\* </ol>

\*

\* @param thumbView The thumbnail view to zoom in.

\* @param imageResId The high-resolution version of the image represented by the thumbnail.

\*/

private void zoomImageFromThumb(final View thumbView, int imageResId) {

// If there's an animation in progress, cancel it immediately and proceed with this one.

if (mCurrentAnimator != null) {

mCurrentAnimator.cancel();

}

// Load the high-resolution "zoomed-in" image.

final ImageView expandedImageView = (ImageView) findViewById(R.id.expanded\_image);

expandedImageView.setImageResource(imageResId);

// Calculate the starting and ending bounds for the zoomed-in image. This step

// involves lots of math. Yay, math.

final Rect startBounds = new Rect();

final Rect finalBounds = new Rect();

final Point globalOffset = new Point();

// The start bounds are the global visible rectangle of the thumbnail, and the

// final bounds are the global visible rectangle of the container view. Also

// set the container view's offset as the origin for the bounds, since that's

// the origin for the positioning animation properties (X, Y).

thumbView.getGlobalVisibleRect(startBounds);

findViewById(R.id.container).getGlobalVisibleRect(finalBounds, globalOffset);

startBounds.offset(-globalOffset.x, -globalOffset.y);

finalBounds.offset(-globalOffset.x, -globalOffset.y);

// Adjust the start bounds to be the same aspect ratio as the final bounds using the

// "center crop" technique. This prevents undesirable stretching during the animation.

// Also calculate the start scaling factor (the end scaling factor is always 1.0).

float startScale;

if ((float) finalBounds.width() / finalBounds.height()

> (float) startBounds.width() / startBounds.height()) {

// Extend start bounds horizontally

startScale = (float) startBounds.height() / finalBounds.height();

float startWidth = startScale \* finalBounds.width();

float deltaWidth = (startWidth - startBounds.width()) / 2;

startBounds.left -= deltaWidth;

startBounds.right += deltaWidth;

} else {

// Extend start bounds vertically

startScale = (float) startBounds.width() / finalBounds.width();

float startHeight = startScale \* finalBounds.height();

float deltaHeight = (startHeight - startBounds.height()) / 2;

startBounds.top -= deltaHeight;

startBounds.bottom += deltaHeight;

}

// Hide the thumbnail and show the zoomed-in view. When the animation begins,

// it will position the zoomed-in view in the place of the thumbnail.

thumbView.setAlpha(0f);

expandedImageView.setVisibility(View.VISIBLE);

// Set the pivot point for SCALE\_X and SCALE\_Y transformations to the top-left corner of

// the zoomed-in view (the default is the center of the view).

expandedImageView.setPivotX(0f);

expandedImageView.setPivotY(0f);

// Construct and run the parallel animation of the four translation and scale properties

// (X, Y, SCALE\_X, and SCALE\_Y).

AnimatorSet set = new AnimatorSet();

set

.play(ObjectAnimator.ofFloat(expandedImageView, View.X, startBounds.left,

finalBounds.left))

.with(ObjectAnimator.ofFloat(expandedImageView, View.Y, startBounds.top,

finalBounds.top))

.with(ObjectAnimator.ofFloat(expandedImageView, View.SCALE\_X, startScale, 1f))

.with(ObjectAnimator.ofFloat(expandedImageView, View.SCALE\_Y, startScale, 1f));

set.setDuration(mShortAnimationDuration);

set.setInterpolator(new DecelerateInterpolator());

set.addListener(new AnimatorListenerAdapter() {

@Override

public void onAnimationEnd(Animator animation) {

mCurrentAnimator = null;

}

@Override

public void onAnimationCancel(Animator animation) {

mCurrentAnimator = null;

}

});

set.start();

mCurrentAnimator = set;

// Upon clicking the zoomed-in image, it should zoom back down to the original bounds

// and show the thumbnail instead of the expanded image.

final float startScaleFinal = startScale;

expandedImageView.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

if (mCurrentAnimator != null) {

mCurrentAnimator.cancel();

}

// Animate the four positioning/sizing properties in parallel, back to their

// original values.

AnimatorSet set = new AnimatorSet();

set

.play(ObjectAnimator.ofFloat(expandedImageView, View.X, startBounds.left))

.with(ObjectAnimator.ofFloat(expandedImageView, View.Y, startBounds.top))

.with(ObjectAnimator

.ofFloat(expandedImageView, View.SCALE\_X, startScaleFinal))

.with(ObjectAnimator

.ofFloat(expandedImageView, View.SCALE\_Y, startScaleFinal));

set.setDuration(mShortAnimationDuration);

set.setInterpolator(new DecelerateInterpolator());

set.addListener(new AnimatorListenerAdapter() {

@Override

public void onAnimationEnd(Animator animation) {

thumbView.setAlpha(1f);

expandedImageView.setVisibility(View.GONE);

mCurrentAnimator = null;

}

@Override

public void onAnimationCancel(Animator animation) {

thumbView.setAlpha(1f);

expandedImageView.setVisibility(View.GONE);

mCurrentAnimator = null;

}

});

set.start();

mCurrentAnimator = set;

}

});

}

}