/\*

\* 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.app.Activity;

import android.content.Intent;

import android.os.Bundle;

import android.support.v4.app.NavUtils;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

/\*\*

\* This sample demonstrates cross-fading between two overlapping views.

\*

\* <p>In this sample, the two overlapping views are a loading indicator and some text content. The

\* active view is toggled by touching the toggle button in the action bar. In real-world

\* applications, this toggle would occur as soon as content was available. Note that if content is

\* immediately available, a loading spinner shouldn't be presented and there should be no

\* animation.</p>

\*/

public class CrossfadeActivity extends Activity {

/\*\*

\* The flag indicating whether content is loaded (text is shown) or not (loading spinner is

\* shown).

\*/

private boolean mContentLoaded;

/\*\*

\* The view (or view group) containing the content. This is one of two overlapping views.

\*/

private View mContentView;

/\*\*

\* The view containing the loading indicator. This is the other of two overlapping views.

\*/

private View mLoadingView;

/\*\*

\* 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\_crossfade);

mContentView = findViewById(R.id.content);

mLoadingView = findViewById(R.id.loading\_spinner);

// Initially hide the content view.

mContentView.setVisibility(View.GONE);

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

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

}

@Override

public boolean onCreateOptionsMenu(Menu menu) {

super.onCreateOptionsMenu(menu);

getMenuInflater().inflate(R.menu.activity\_crossfade, menu);

return true;

}

@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;

case R.id.action\_toggle:

// Toggle whether content is loaded.

mContentLoaded = !mContentLoaded;

showContentOrLoadingIndicator(mContentLoaded);

return true;

}

return super.onOptionsItemSelected(item);

}

/\*\*

\* Cross-fades between {@link #mContentView} and {@link #mLoadingView}.

\*/

private void showContentOrLoadingIndicator(boolean contentLoaded) {

// Decide which view to hide and which to show.

final View showView = contentLoaded ? mContentView : mLoadingView;

final View hideView = contentLoaded ? mLoadingView : mContentView;

// Set the "show" view to 0% opacity but visible, so that it is visible

// (but fully transparent) during the animation.

showView.setAlpha(0f);

showView.setVisibility(View.VISIBLE);

// Animate the "show" view to 100% opacity, and clear any animation listener set on

// the view. Remember that listeners are not limited to the specific animation

// describes in the chained method calls. Listeners are set on the

// ViewPropertyAnimator object for the view, which persists across several

// animations.

showView.animate()

.alpha(1f)

.setDuration(mShortAnimationDuration)

.setListener(null);

// Animate the "hide" view to 0% opacity. After the animation ends, set its visibility

// to GONE as an optimization step (it won't participate in layout passes, etc.)

hideView.animate()

.alpha(0f)

.setDuration(mShortAnimationDuration)

.setListener(new AnimatorListenerAdapter() {

@Override

public void onAnimationEnd(Animator animation) {

hideView.setVisibility(View.GONE);

}

});

}

}