Simple animation comes in four "flavors" in android
- Drawable Animation (aka Frame Animation)
- Layout Animation
- View Animation (related to Layout)
- Property animation (Android 3.0+, becoming preferred)

Android 4.4 added a Transitions Framework that makes using some of the above easier

Drawable Animation - load a series of drawables and show them in some order
- Can specify duration on each drawable
- (Similar to film movie projectors, film is typically 24 frames/sec)
- AnimationDrawable class or XML <animation-list> works well.
- Modes:
  - One shot -- run the animation once
  - Continual -- does it over and over
Sample code: (SampleFrameAnimation from ProAndroid4)

```java
ImageView imgView = (ImageView)findViewById(R.id.imageView);
imgView.setVisibility(ImageView.VISIBLE);
imgView.setBackgroundResource(R.drawable.frame_animation);

AnimationDrawable frameAnimation = (AnimationDrawable) imgView.getBackground();
frameAnimation.start(); // or .stop()
```

- Test for "running"
  - frameAnimation.isRunning()
Frame Animation Demo Program

Most of code is setting up the views
- code on previous slide plus XML animation-list is all that is needed

```xml
<animation-list android:id="selected" android:oneshot="false">
  <item android:drawable="@drawable/wheel0" android:duration="50" />
  <item android:drawable="@drawable/wheel1" android:duration="50" />
  <item android:drawable="@drawable/wheel2" android:duration="50" />
  <item android:drawable="@drawable/wheel3" android:duration="50" />
  <item android:drawable="@drawable/wheel4" android:duration="50" />
  <item android:drawable="@drawable/wheel5" android:duration="50" />
</animation-list>
```

Methods in AnimationDrawable Class
- `addFrame(Drawable frame, int duration)` -- code to add a frame
  - `duration` in milliseconds
- `Drawable getFrame(int index)` -- gets a frame from the list
- `int getDuration(int i)` -- gets the duration of the frame
- `boolean setVisible(boolean visible, boolean restart)`
  - `restart` true => start at first frame again
- `void start()` and `void stop()`
Layout Animation

Can use this to do tweened animation on Views

- "simple" transformations of:
  - position - x, y (view moves, "button" touchables don’t)
  - size - Whole view changes size
  - rotation - Whole view is rotated
  - alpha - fade in / fade out (transparency of view)

SampleLayoutAnimation demo program (from ProAndroid4)

- All animation done via XML
- Animation on the List View
- Results are on the list elements (during layout)
View Animation

- Similar animation transformations, position, size, rotation, alpha

- View class methods
  - setPivotX(float), setPivotY(float)
  - setRotation(float), setRotationX(float), setRotationY(float)
  - scaleX(float), scaleY(float)
  - setTranslationX(float), setTranslationY(float)
  - setAlpha(float)
  - setAnimation(Animation) -- sets an animation on the view
Animation class

android.view.animation.Animation -- extend to use

- Sets view parameters over time ....
  - public class MyAnimation extends Animation ...
  - View v = findViewById(R.id.xxxx);
  - v.startAnimation(new MyAnimation());

- Animation class is where the action is located.
- Examples program (View Animation from book with my tweaks)
  - SampleViewAnimation.tgz
  - ViewAnimation2.java (easiest) -- "two" from menu
    - 2.5 second animation
    - run scale of transform from 0 to 1
    - applyTransformation() -- called multiple times
    - sets up the transformation matrix with proper values
    - framework uses the transformation matrix to "show view"
View Animation (Page 2)

- ViewAnimation3.java and ViewAnimation.java essentially the same
  - 2.5 second animation
  - different constructors
  - scale and translate
    - preTranslate/postTranslate -- before scale

- ViewAnimation1.java -- most complex
  - Uses a "Camera" to provide effects
  - "Moves" the Camera in 3d
  - "Moves" the Camera around the view
  - Renders the camera view
    - Actually, just makes a transformation matrix
    - Does normal rendering of view modified by matrix
Property animation

Animation of anything that has a property ....

- Much more general method than view or frame animation
- Can define the following characteristics of an animation
  - Duration -- length of time for the animation
  - Time interpolation -- how fast things change in relation to time
  - Repeat behavior -- once, multiple, continuous, reverse, ...
  - Animator sets -- collections of animations that need to run
    - at the same time
    - sequentially
    - other timing constraints
  - Frame refresh delay -- default 10ms

- This was used in the fragments example for animating frames
- Need to let this be "further reading" for this class
- Projects for Android Studio found at /home/phil/public/csci412
  - SampleXAnimation.tgz, X = Frame, Layout, Property, View
  - ShakespeareInstrumented.tgz -- animation with fragments
- Extract a .tgz file: tar -xzf file.tgz
Transitions framework was introduced in Android 4.4 to help make animation "easier"

Animation of layout changes via this framework

Along with the previous animations, this adds "Scenes"

Uses Layout and View animation to do the work

Basic idea: Starting layout transition to ending layout

Group-level animations: Apply one or more animation effects to all views

Built-in animations: Adds a collection of common effects

Resource file support: layouts and animations from resource files

TransitionManager class -- Controls the transition

TransitionDemo app -- simple single button modification

Use of beginDelayedTransition()

Use of code to control transition, change constraints on the fly

Scene idea:

Use XML or code to produce two layouts

Idea of a root container (view)

Scene is a layout to go into the container
Transitions between scenes are done by the TransitionManager

Scene scene1 = Scene.getSceneForLayout(rootContainer, R.layout.scene1_layout, this)

There are other ways to create scenes.

Show the scene with no transition: scene1.enter()

Transition (using the default): TransitionManager.go(scene2)

Scenes can be created by viewgroups: Scene2 = Scene(viewGroup1)

Transitions are done by interpolaters (doing the tweening)

AccelerateInterpolator, AccelerateDecelerateInterpolator, DecelerateInterpolator

AnticipateInterpolator, AnticipateOvershootInterpolator, OvershootInterpolator

BounceInterpolator

CycleInterpolator

LinerInterpolator

Custom transitions may be defined

XML based: fade and change bounds

Specify a transition set, parallel or sequential

Can be done in code

SceneTransitions demo has a simple two scene transition

As shown, modified from books code: fixed a bug, timings changed