Material Design CoP Content

Development Concepts

1. View Clipping
    a. [Info slide] **Clipping**
        i. Refers to the act of re-shaping a view to its outline.
        ii. This means we can take a rectangular image and clip it into a circular shape.
        iii. Lets begin with a rectangular image.
    b. [Image slide]
        i. Breaking Bad – Walter White Meme
    c. [Title slide]
        i. That image needs some elements removed.
        ii. We can remove these by clipping.
    d. [Image slide]
        i. Interstitial w/ Walter White FAB
    e. [Title slide]
        i. The Code
    f. [Image slide]
        i. ImageView layout xml with src pointed to Walter White Image and background pointed to the oval shape drawable
    g. [Image slide]
        i. Oval shape drawable
    h. [Image slide]
        i. Java code to clip the ImageView to shape drawable
    i. [Info slide] **Summary**
        i. Clipping isn't hard but it does come with a caveat, it's expensive.
        ii. Note: rectangle, circle and round rectangle are the only shapes that support outline clipping.
2. Activity Transition
    a. [Info slide] **Three Types**
        i. Enter
        ii. Exit
        iii. Shared Element
    b. [Info slide] **Enter \ Exit Transitions**
        i. Explode

    ii. Slide
    iii. Fade
c. [Info slide] **Enter \ Exit Transitions**
    i. Can be specified in code with transition objects.
    ii. These objects can be custom or use a pre-existing transition
d. [Info slide] **Shared Element**
    i. changeBounds
    ii. changeClipBounds
    iii. changeTransform
    iv. moveImage
e. [Info slide] **Example Transitions**
    i. BUT WAIT! We need to enable transitions, either in code or within our style.
    ii. Preferred method, enable it within the projects style.xml
f. [Image slide]
    i. style.xml
g. [Title slide]
    i. Exit Example
h. [Image slide]
    i. Exit Example Code Image
i. [Info slide] **What the what?**
    i. What was that setAllowExitTransitionOverlap(true) method for?
    ii. To start an exit transition as soon as possible. This lets you have more dramatic enter transitions.
j. [Title slide]
    i. Enter Example
k. [Image slide]
    i. Enter Example Code Image
l. [Info slide] **What the what?**
    i. What was that setAllowEnterTransitionOverlap(true) method for?
    ii. To start an enter transition as soon as possible. This lets you have more dramatic enter transitions.
m. [Title slide]
    i. Shared Element Example
n. [Title slide]

        i. Caller:
    o. [Image slide]
        i. Code showing the caller assigning view names to the title and info section of it's 'shared' view
    p. [Title slide]
        i. Callee:
    q. [Image slide]
        i. Layout showing the callee assigning view names in the layout xml for it's 'shared' views
    r. [Title slide]
        i. Kick off:
    s. [Image slide]
        i. Code showing the caller linking the shared view names and starting the activity with ActivityOptions (the animation)
    t. [Title slide]
        i. Result:
    u. [Video slide]
        i. Video showing the resulting Activity Transition
    v. [Information slide] **Things to Note**
        i. If the current activity MUST finish but a transition is being performed, call activity.finishAfterTransition() vs. activity.finish()
        ii. This way the transition animation is not choppy or interrupted.
3. View Animation
    a. [Info slide] **Let's be Clear**
        i. Not to be confused with Activity Transition, Ripple feedback, or State Changes (transition z).
        ii. View animation deals with displaying views in a colorful fashion
    b. [Info slide] **Two General Methods:**
        i. Reveal Effect
        ii. Curved Motion
    c. [Info slide] **Reveal Effect**
        i. Done with the new ViewAnimationUtils class. This class has only one method, createCircularReveal(), which as the name implies, creates a circular reveal for the view.

    ii. Note: This is a preferred method for views with clipped outlines.
d. [Video slide]
    i. Video showing the playground list and it's cardviews revealing while scrolling down.
e. [Image slide]
    i. Image showing the code when scrolling up in that list
f. [Image slide]
    i. Image showing the code when scrolling down in that list
g. [Info slide] **Curved Motion**
    i. Animations in material design rely on curves for time interpolation and spatial movement patterns. New APIs enable you to define custom timing curves and curved motion patterns for animations.
h. [Info slide] **Old Interpolators**
    i. We already have accelerate / decelerate interpolators; interpolators where the rate of change starts and ends slowly but accelerates through the middle.
i. [Video slide]
    i. Video showing the accelerate / decelerate interpolator when scrolling the playground list view
j. [Info slide] **New Interpolator!**
    i. PathInterpolator. Similar but different.
    ii. Based on a Bezier curve on a Path object.
k. [Video slide]
    i. Video showing the path interpolator when scrolling the playground list view
l. [Video slide]
    i. Video highlighting the differences between path and linear (accelerate / decelerate) a little better
m. [Image slide]
    i. Image showing the path interpolator layout xml
n. [Image slide]
    i. Image showing the code to do the same thing the layout xml is doing
o. [Image slide]

        i. Image showing the ImageView referencing the interpolator
    p. [Image slide]
        i. Image showing the java code referencing the interpolator
4. Elevation
    a. [Info slide] **Shadows**
        i. Views can now cast shadows based on differences in elevation; their Z plane property.
        ii. As it stands now, shadows are only cast on the Z = 0 plane.
    b. [Info slide] **Z Movement Animation**
        i. Translation Z is another property that can be used to animate changes in an elements Z plane.
        ii. Similar to existing methods for Translation X and Translation Y; which both exist now.
    c. [Info slide] **Z Movement Animation**
        i. Translation Z adjusts an elements depth relative to its current depth. Confused? Don't be!
        ii. Image we have a view with static elevation = 1. If we translate that views z plane, we're essentially moving it towards or away from us.
    d. [Image slide]
        i. Image showing values in the dimens.xml file to be used for translation in z.
    e. [Image slide]
        i. Image showing the object animator drawable xml which references the dimens.xml values for translation in z.
    f. [Image slide]
        i. Image showing ImageButton referencing the state list animator
    g. [Video slide]
        i. Video showing the elevation changes for the elevation cardview in the playground.
    h. [Title slide]
        i. This can also be achieved in code.
    i. [Image slide]
        i. Image showing the code to achieve the same affect.

j. [Info slide] **Changing Depth**
    i. Views can optimize the OS handled logic to demonstrate more actions to the user.
k. [Info slide] **Side Note**
    i. By clipping a view to its outline we ensure its shadow is correct during elevation modifications.

5. Ripple
    a. [Info slide] **Touch Feedback**
        i. Until now we've had a general two ways to represent feedback.
            1. Depressed Buttons
            2. Non-Depressed Buttons
    b. [Image slide]
        i. Image showing ripples in water
    c. [Info slide] **Ripple**
        i. Allows us as developers, to convey to the user the UI has registered a touch event.
        ii. Can be used in combination with button depression.
    d. [Info slide] **Buttons & EditText**
        i. Get this ripple affect for free!
        ii. You can specify the default touch feedback color in your applications style.xml
    e. [Image slide]
        i. Image showing default color highlight in style.xml
    f. [Video slide]
        i. Video showing default rippling for buttons
    g. [Info slide] **U.G.L.Y?!**
        i. Why isn't that prettier?
        ii. We forgot to apply the ability for the buttons to match their surroundings.
        iii. By doing so, we allow the ripple to 'flow' through the UI element.
        iv. There are two options for doing this.
    h. [Image slide]
        i. Image showing two buttons implementing the two different styles for ripple affect
    i. [Video slide]
        i. Video showing the ripple affect for the two implementations

j. [Title slide]
    i. Better!!! What else do we get for free?
k. [Video slide]
    i. Video showing the EditText highlight affect
l. [Video slide]
    i. Video showing the side menu click ripple affect
m. [Info slide] **Custom Views**
    i. But what about custom views? How do we allow them to use this ripple affect?
    ii. The preferred approach is to do it within the XML.
n. [Image slide]
    i. Image showing the ImageButton referencing a drawable as its background
o. [Image slide]
    i. Image showing the drawable's layout xml, which includes a ripple affect
p. [Image slide]
    i. Image showing the drawables background shape, oval, and color
q. [Video slide]
    i. Video showing the FAB ripple affect.
r. [Info slide] **Summary**
    i. Easy does it.
    ii. Easily done.

6. CardView
    a. [Info slide] **extends FrameLayout**
        i. Providing you the ability to show information inside cards that have a consistent look on any app.
    b. [Image slide]
        i. Image showing the layout xml referencing a CardView
    c. [Image slide]
        i. Image showing the resulting ui from the layout xml
    d. [Info slide] **Features**
        i. Customizable corner radius
        ii. Adjustable elevation with appropriate shadow outline
        iii. Even background color adjustments!
    e. [Title slide]
        i. It's in the appcompate v7 support library