BCS 371 Mobile Application Development I

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

Animation

- AnimatedVisibility
- Value-based animations (animate*AsState)
  - Color changes (animateColorAsState)
  - Content size changes (animateDpAsState)
  - Any float value changes (animateFloatAsState)
- Run function when animation finishes
- NavHost – Navigating between screens

Animation

AnimatedVisibility

- Perform an animation when making a composable visible or invisible.
- Can run an animation when entering or exiting.
- Can control the speed of the animation.
- Can control which direction the composable comes on screen from.

- Here is a link with descriptions of various parameters to use: https://developer.android.com/develop/ui/compose/animation/composables-modifiers

AnimatedVisibility

AnimatedVisibility Example - enter

```
var isVisible by remember { mutableStateOf(false) }
Column(modifier = Modifier.padding(innerPadding)) {
    Button(
        onClick = { isVisible = !isVisible }
    ) {
        Text("Show/Hide Box")
    }

    AnimatedVisibility(
        visible = isVisible,
        enter = slideInHorizontally(),
        modifier = Modifier.fillMaxWidth().weight(1f)
    ) {
        Box(modifier = Modifier.background(Color.Blue))
    }
} // end - Column
```

Assumes the Column is inside a Scaffold with a parameter named innerPadding (makes content start below toolbar)

Button toggles the value of the isVisible variable

When the value of isVisible changes a recomposition occurs for the AnimatedVisibility composable

Set how it enters (becomes visible). This will enter horizontally.

1f weight will cause it to fill the open space in the column since no siblings in the column specified a weight (the Button did not specify a weight)

The Box is a child of AnimatedVisibility. The Box's visibility will be animated.

AnimatedVisibility - enter

AnimatedVisibility Example - exit

```
var isVisible by remember { mutableStateOf(false) }
Column(modifier = Modifier.padding(innerPadding)) {
    Button(
        onClick = { isVisible = !isVisible }
    ) {
        Text("Show/Hide Box")
    }

    AnimatedVisibility(
        visible = isVisible,
        exit = slideOutHorizontally(),
        modifier = Modifier.fillMaxWidth().weight(1f)
    ) {
        Box(modifier = Modifier.background(Color.Blue))
    }
} // end - Column
```

Set how it exits (becomes invisible).
This will exit horizontally.

AnimatedVisibility - exit

AnimatedVisibility Options

- **Set duration of animation.** Tween creates an animation specification (the tween name comes from "between" because it is animating between values). The value passed to tween here is the duration in milliseconds (2 seconds).

enter = slideInHorizontally(tween(2000))

- **Slide in vertically.**

enter = slideInVertically()

- **Slide out vertically.**

exit = slideOutVertically()

AnimatedVisibility Options

Value-based Animations

- Perform an animation when a single value changes.
- For example, animate a color value change.

- animate*AsState – There are a set of functions where the * is replaced with some property. For example:
  - animateColorAsState
  - animateDpAsState
  - animateFloatAsState

Value-based Animations

animateColorAsState

- Performs an animation when the color value changes.

```
var color by remember { mutableStateOf(Color.Green) }

val colorAsState: Color by animateColorAsState(
    color,
    animationSpec = tween(2000)
)

// Use colorAsState in a composable
Box(
    modifier = Modifier
        .background(colorAsState)
        .size(64.dp)
)

// Set the value of color somewhere else (for example on a button click)
color = Red
```

The colorAsState animation uses the color variable value. When the value of color changes the animation is triggered.

Animation will take 2 seconds

Box uses the color value in the colorAsState animation. It will recompose each time the value changes. As the animation progresses the color value will keep changing and it will keep recomposing.

The animation will execute when the value of color changes

animateColorAsState

animateDpAsState

- Performs an animation when the dp value changes.

```
var dp by remember { mutableStateOf(64.dp) }
val dpAsState by animateDpAsState(
    dp,
    animationSpec = tween(durationMillis = 2000)
)

// Use dpAsState in a composable
Box(
    modifier = Modifier
        .background(Blue)
        .size(dpAsState)
)

// Set the value of dp somewhere else (for example on a button click)
dp = 128.dp
```

The dpAsState animation uses the dp variable value

Animation will take 2 seconds

Box uses the dp value in the dpAsState animation

The animation will execute when the value of dp changes

animateDpAsState

animateFloatAsState

- Performs an animation when the float value changes.
- The rotate method is being used (takes a float).

```
var rotateAngle by remember { mutableStateOf(0f) }
val rotateAngleAsState by animateFloatAsState(
    rotateAngle,
    animationSpec = tween(durationMillis = 2000)
)

// Use rotateAngleAsState in a composable
Box(
    modifier = Modifier
        .rotate(rotateAngleAsState)
        .background(Blue)
)

// Set the value of rotateAngle somewhere else (for example on a button click)
rotateAngle = 45f
```

The animateFloatAsState animation uses the dp variable value

Animation will take 2 seconds

Box uses the rotateAngle value in the dpAsState animation. IMPORTANT! Make sure to call rotate first in the modifier

The animation will execute when the value of rotateAngle changes

animateFloatAsState

Run Function When Animation Finishes

```
val animationFinished: (Color) -> Unit = {
    Toast.makeText(context, "Animation finished", Toast.LENGTH_SHORT).show()
}

var color by remember { mutableStateOf(Blue) }
val colorAsState by animateColorAsState(
    color,
    animationSpec = tween(
        durationMillis = 2000,
        easing = LinearOutSlowInEasing
    ),
    label = "color animation",
    finishedListener = animationFinished
)
```

Running Function When Animation Finishes

NavHost – Animating Between Screens

- Show an animation when navigating between screens.
- Use the enterTransition and exitTransition parameters.

```
NavHost(navController=navController,
    startDestination = "MainScreen",
    enterTransition = {
        slideIntoContainer(AnimatedContentTransitionScope.SlideDirection.Start,
        tween(500))
    },
    exitTransition = {
        slideOutOfContainer(AnimatedContentTransitionScope.SlideDirection.Start,
        tween(500))
    },
    modifier = modifier)
{
    composable(route="MainScreen") {
        MainScreen(navController)
    }
    composable(route="OtherScreen") {
        OtherScreen(navController)
    }
} // end - NavHost
```

enterTransition – Animation for going to a screen

exitTransition – Animation for leaving a screen

NavHost – Animating Between Screens

- End of Slides

End of Slides