

Animations

Jetpack Compose

Jakub Zíka
Android Developer at CN Group

Introduction

- What is Jetpack Compose?
 - ▶ Declarative vs imperative
 - ▶ Stateless vs stateful (getter/setter)
 - ▶ No Fragments needed (side-effects)
 - ▶ No XML inflation



View vs Composable

```
private lateinit var contentView: View
private lateinit var loadingView: View
private var shortAnimationDuration: Int = 0
```

```
private fun crossfade() {
    contentView.apply {
        // Set the content view to 0% opacity but visible, so that it is visible
        // (but fully transparent) during the animation.
        alpha = 0f
        visibility = View.VISIBLE

        // Animate the content view to 100% opacity, and clear any animation
        // listener set on the view.
        animate()
            .alpha(1f)
            .setDuration(shortAnimationDuration.toLong())
            .setListener(null)
    }
    // Animate the loading 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.)
    loadingView.animate()
        .alpha(0f)
        .setDuration(shortAnimationDuration.toLong())
        .setListener(object : AnimatorListenerAdapter() {
            override fun onAnimationEnd(animation: Animator) {
                loadingView.visibility = View.GONE
            }
        })
}
```

Crossfade in View (XML)

```
var currentPage by remember { mutableStateOf("A") }
Crossfade(targetState = currentPage) { screen ->
    when (screen) {
        "A" -> Text("Page A")
        "B" -> Text("Page B")
    }
}
```

Crossfade in Composable

API

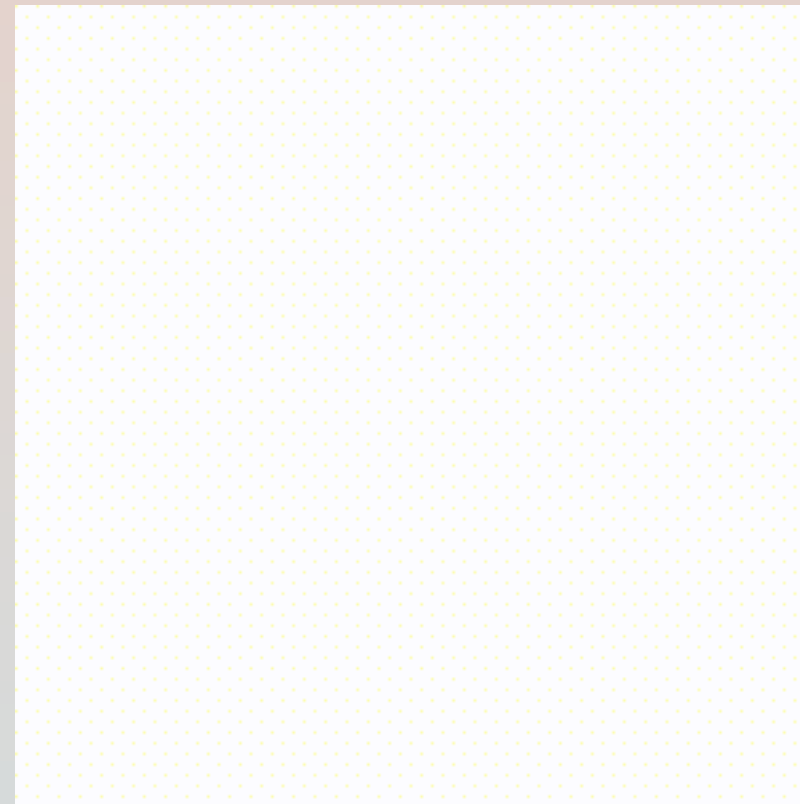
- **High-level API** (defined as Composables)
 - AnimatedVisibility, AnimatedContent, Crossfade
 - animate*AsState (Float, Dp, Color, Int, Offset, Rect, Size, Value)
 - rememberInfiniteTransition, updateTransition
 - Modifier.animateContentSize
- **Low-level APIs** (can be defined outside of Composables)
 - Animation (TargetBasedAnimation, DecayAnimation)
 - Animatable

Animations

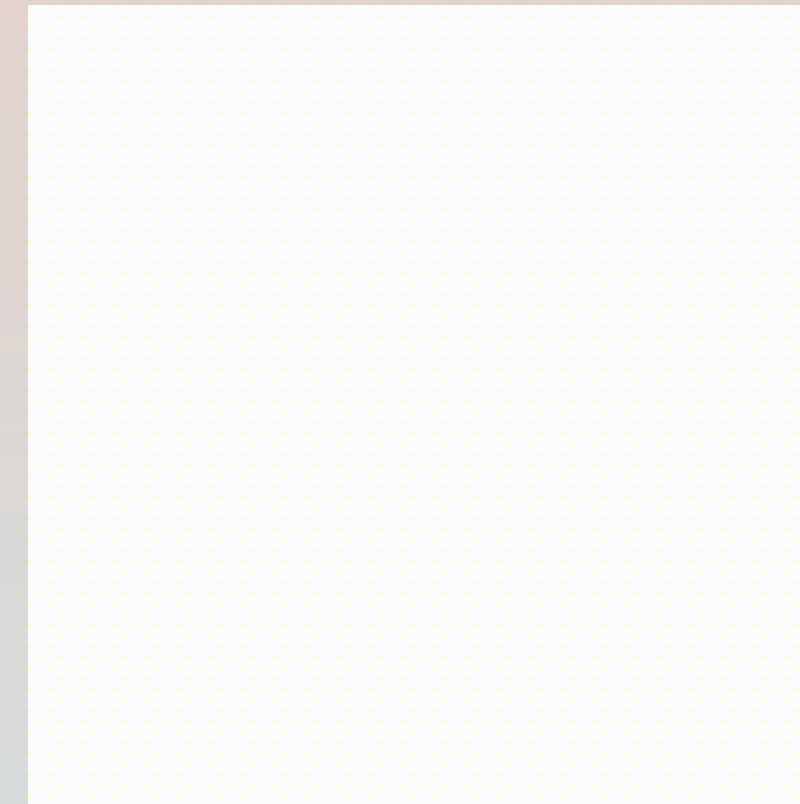
FadeIn



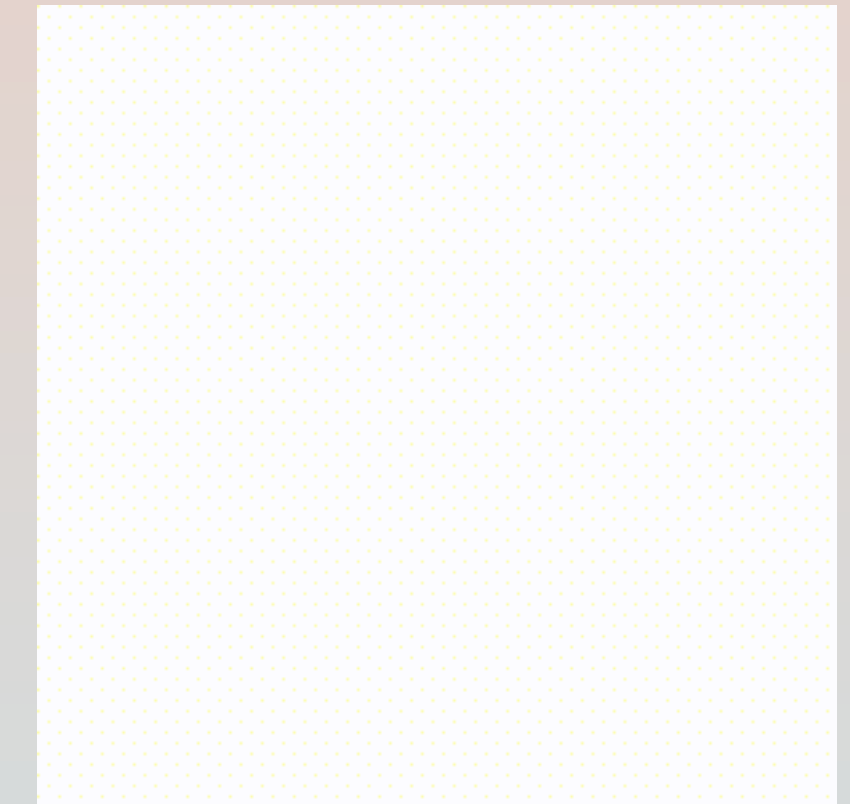
SlideInHorizontally



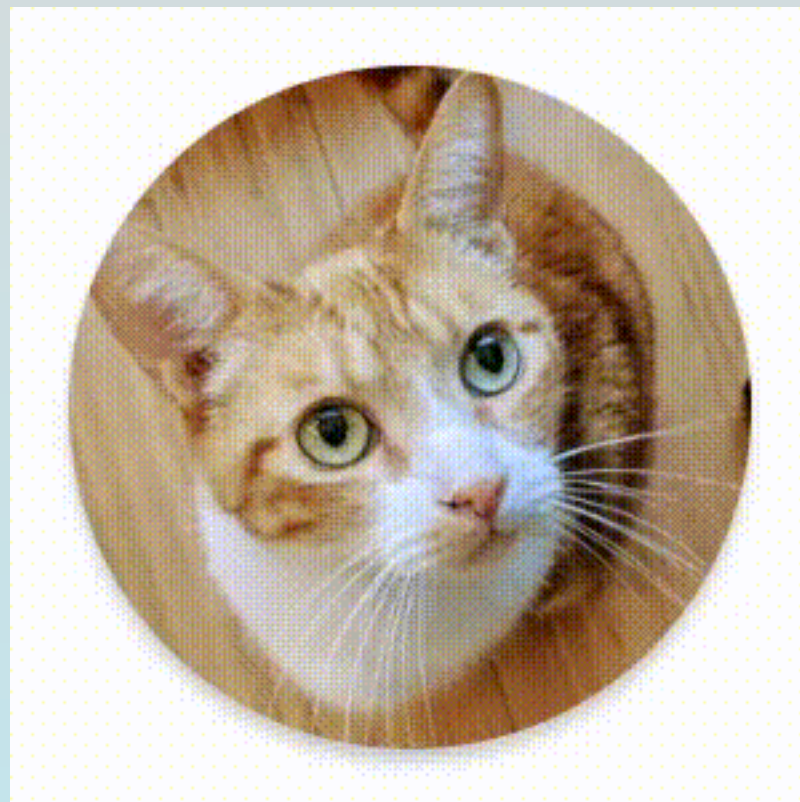
ExpandHorizontally



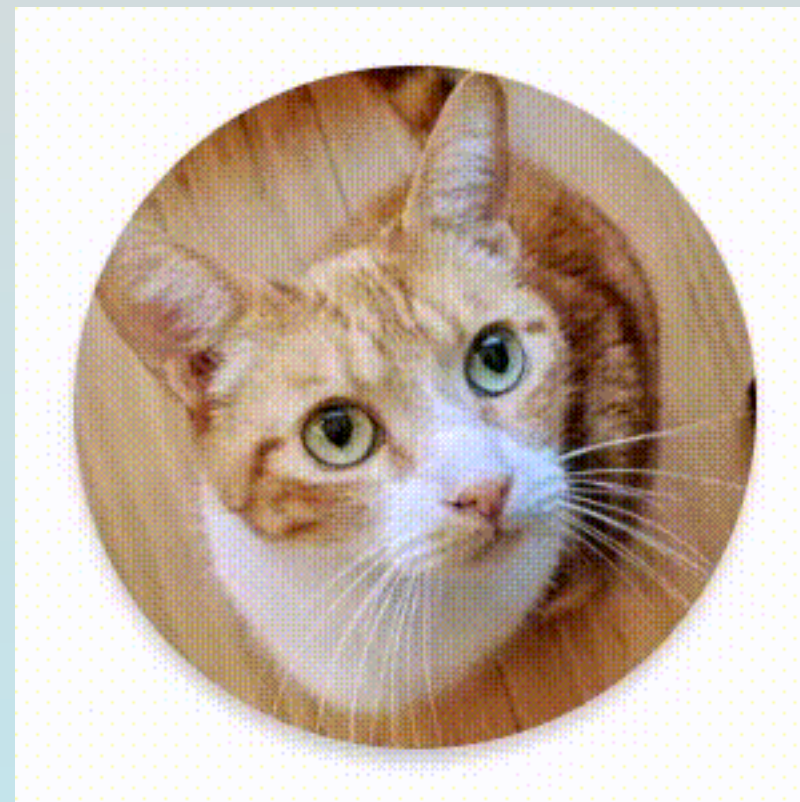
ScaleIn



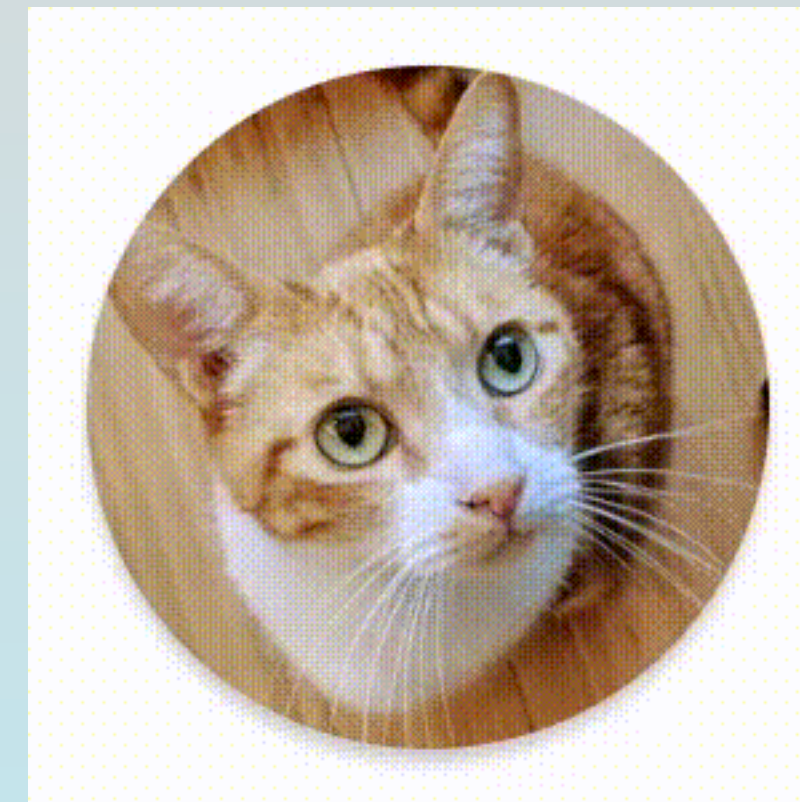
FadeOut



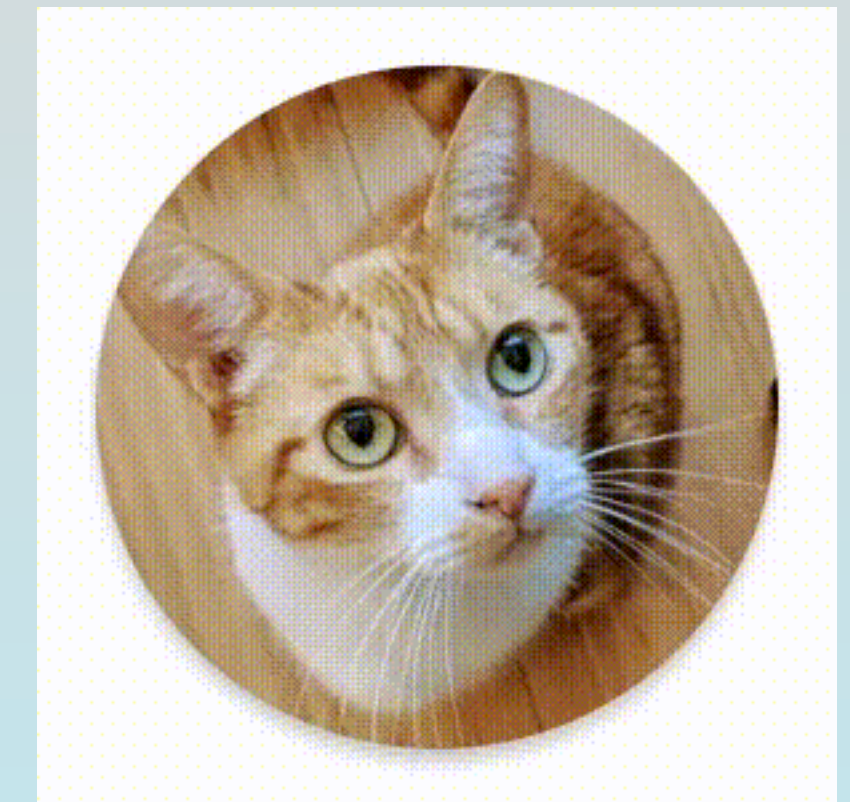
SlideOutHorizontally



ShrinkHorizontally



ScaleOut



ChatGPT



Create for me animation in Jetpack Compose with using `AnimatedContent` where the Icon is moving in spiral.

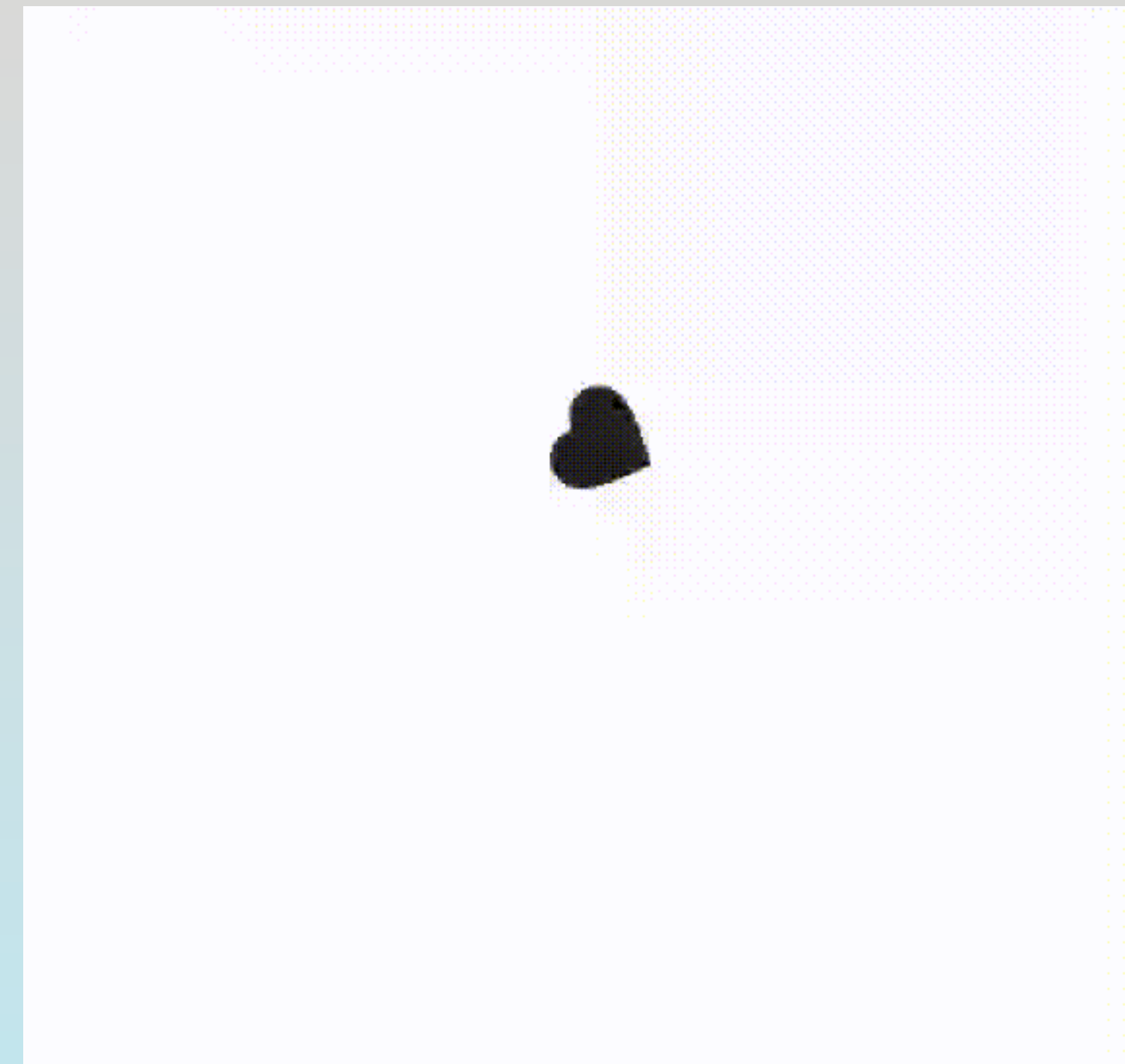


Sure, I can help you create an animation in Jetpack Compose where an Icon moves in a spiral pattern using `AnimatedContent`. Here's some sample code to get you started:

Here's a quick explanation of the code:

- We start by defining a mutable state variable `rotationAngle` that will be used to rotate the Icon.
- We use the `rememberInfiniteTransition()` function to create an `InfiniteTransition` object that will be used to animate the spiral movement of the Icon.
- We use the `animateFloat()` function to define two animations - one for the spiral angle (`spiralAngle`) and one for the spiral radius (`spiralRadius`). Both animations use the `infiniteRepeatable()` function to repeat indefinitely, and they both use a linear easing function to create a smooth movement.
- Inside the `Box` composable, we use the `Icon` composable to display the Icon. We use the `offset()` modifier to move the Icon in a spiral pattern based on the current `spiralAngle` and `spiralRadius` values. We use the `rotate()` modifier to rotate the Icon based on the current `rotationAngle` value.

Note: This animation will start as soon as it's rendered on the screen and will run indefinitely. To stop the animation, you'll need to remove the composable from the screen or conditionally render it based on a variable.



Resources

- Android Animations Documentation (<https://developer.android.com/jetpack/compose/animation>)
- Android Animations Training (<https://developer.android.com/codelabs/jetpack-compose-animation#0>)
- MacGPT (app for MacOS)

