

Lecture #9

Animations

Mobile Applications
Fall 2023

Overview

- Add visual cues about what is going on.
- Useful when the UI changes states.
- Adding a polished look, gives a higher quality look and feel.
- Add motions to the UI.

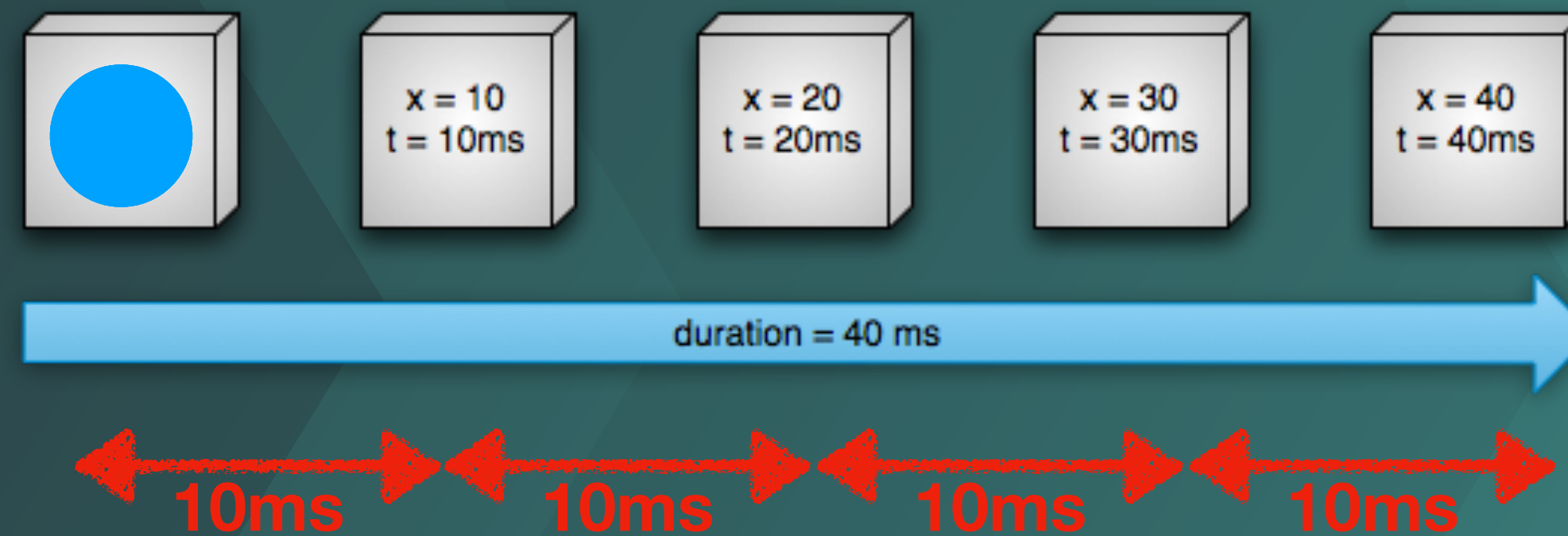


Property Animation

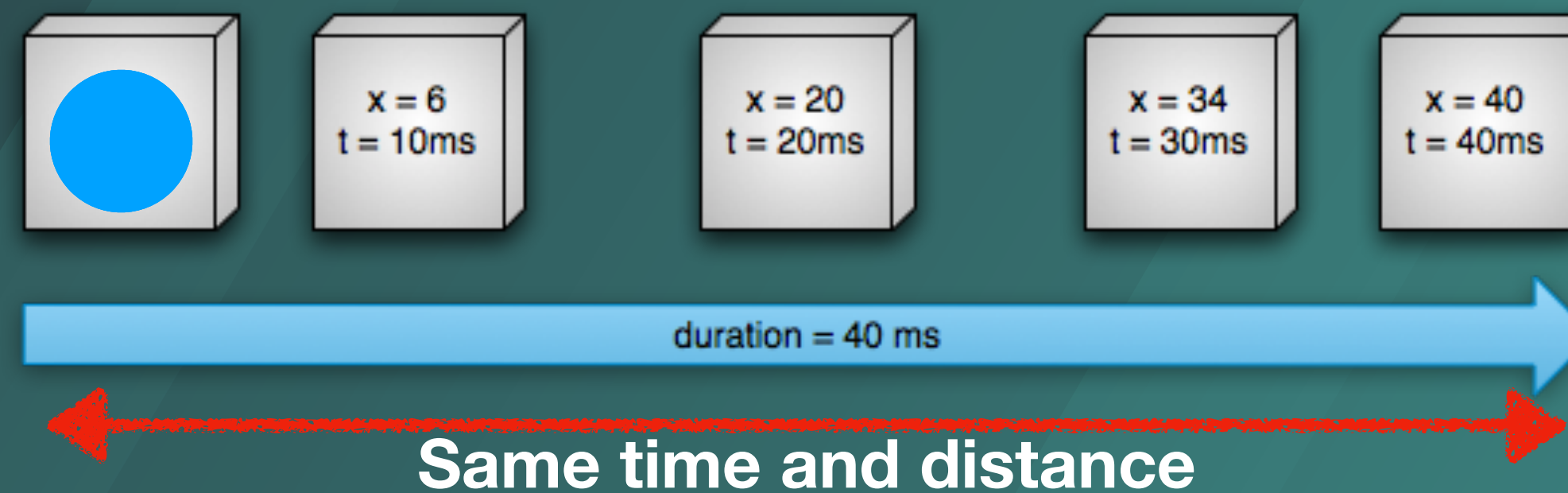
- Robust framework that allows to animate almost anything.
- Defines animation to change any object property over time.
- Characteristics of an animation:
 - Duration. Default length: 300ms.
 - Time interpolation. Defines how the values for the property are calculated.
 - Repeat count and behavior.
 - Animation sets.
 - Frame refresh delay. Default value: 10ms.

How property animation works

Linear animation



Non-linear animation



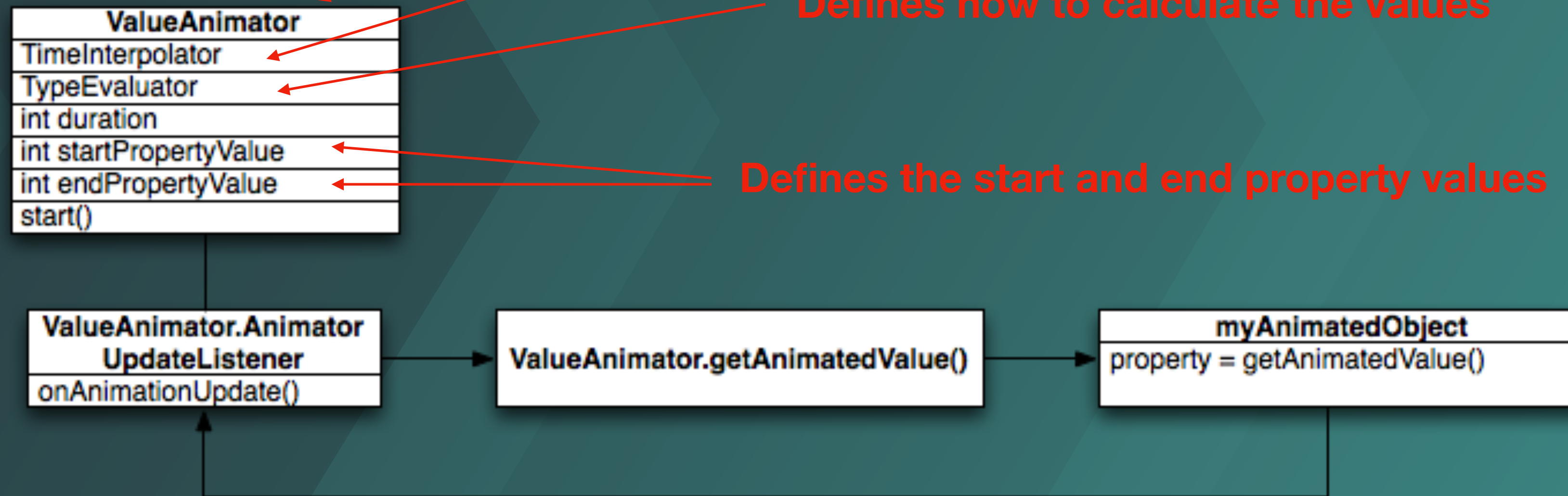
Model

Keeps track of the animation timing

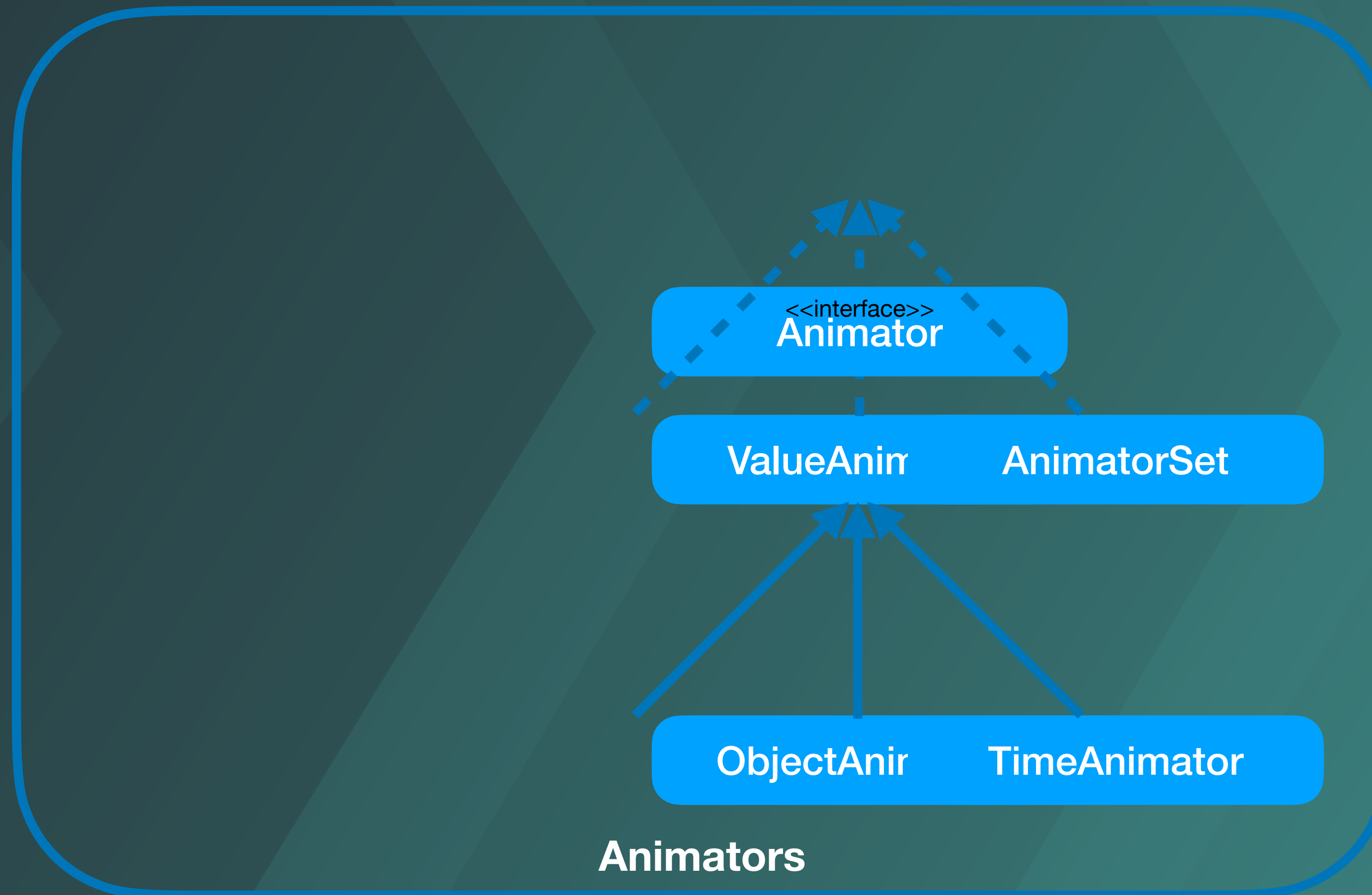
Defines the time interpolation

Defines how to calculate the values

Defines the start and end property values



API



<https://developer.android.com/reference/android/animation/Animator>

API

Animators

IntEvaluator

FloatEvaluator

ArgbEvaluator

IntArrayEvaluator

FloatArrayEvaluator

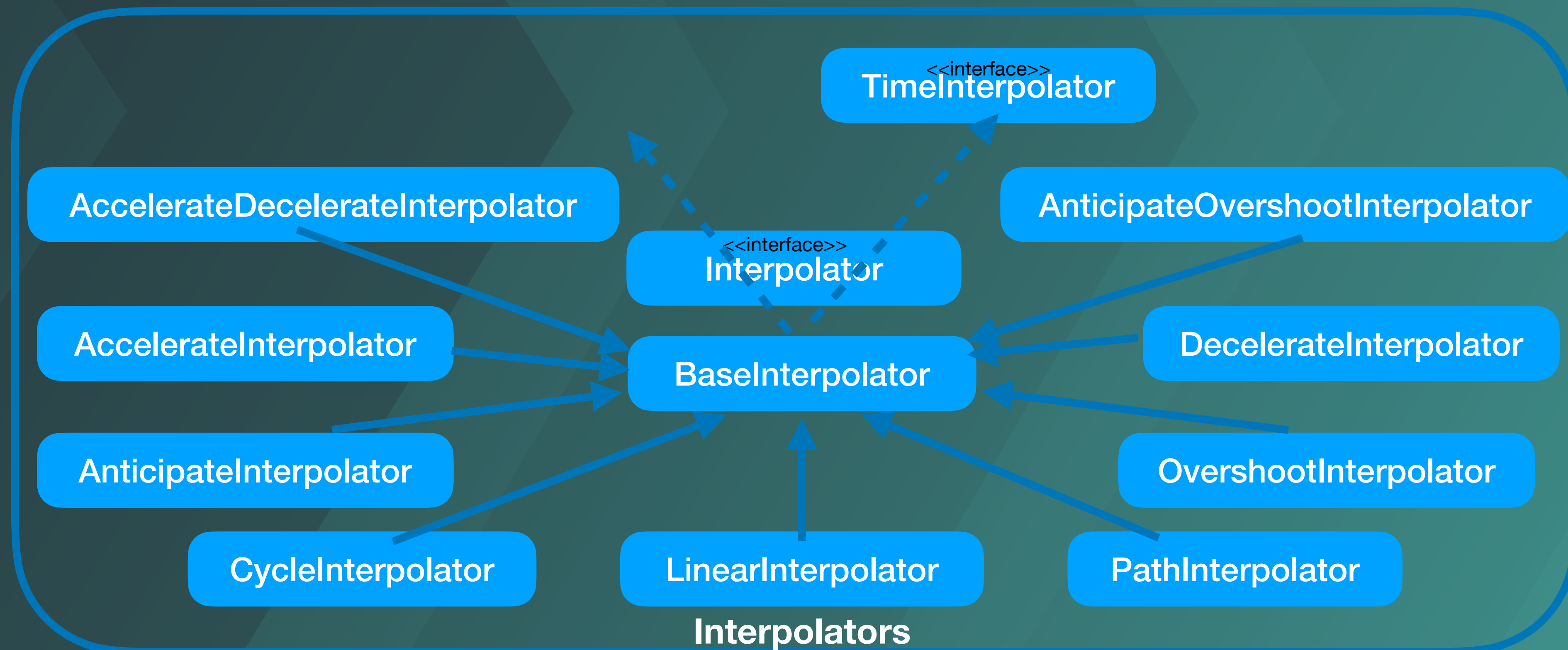
Evaluators

<https://developer.android.com/reference/android/animation/TypeEvaluator>

API

Animators

Evaluators



<https://developer.android.com/reference/android/view/animation/Interpolator>

API

Animators

Evaluators

Interpolators

```
ValueAnimator.ofObject(...).apply {  
    // ...  
    ValueAnimator.ofFloat(0f, 100f).apply {  
        addUpdateListener { updatedAnimation ->  
            duration = 1000  
            // You can use the animated value in a property that uses the  
            // same type as the animation. In this case, you can use the  
            ValueAnimator.ofObject(MyTypeEvaluator(),  
                textView.translationXendPropertyAnimation as Float  
            } duration = 1000  
            //start()  
        }  
    }  
ObjectAnimator.ofFloat(textView, "translationX", 100f).apply {  
    duration = 1000  
    start()  
}
```

Choreograph using an AnimatorSet

```
val bouncer = AnimatorSet().apply {  
    play(bounceAnim).before(squashAnim1)  
    play(squashAnim1).with(squashAnim2)  
    play(squashAnim1).with(stretchAnim1)  
    play(squashAnim1).with(stretchAnim2)  
    play(bounceBackAnim).after(stretchAnim2)  
}
```

```
val fadeAnim = ObjectAnimator.ofFloat(newBall, "alpha", 1f, 0f).apply {  
    duration = 250  
}
```

```
AnimatorSet().apply {  
    play(bouncer).before(fadeAnim)  
    start()  
}
```

Animation Listeners

```
ObjectAnimator.ofFloat(newBall, "alpha", 1f, 0f).apply {  
    duration = 250  
    addListener(object : AnimatorListenerAdapter() {  
        override fun onAnimationEnd(animation: Animator) {  
            balls.remove((animation as ObjectAnimator).target)  
        }  
    })  
}
```

Animate Layout Changes

```
<LinearLayout  
    android:orientation="vertical"  
    android:layout_width="wrap_content"  
    android:layout_height="match_parent"  
    android:id="@+id/verticalContainer"/>  
    android:animateLayoutChanges="true" />
```


Animate View State Changes

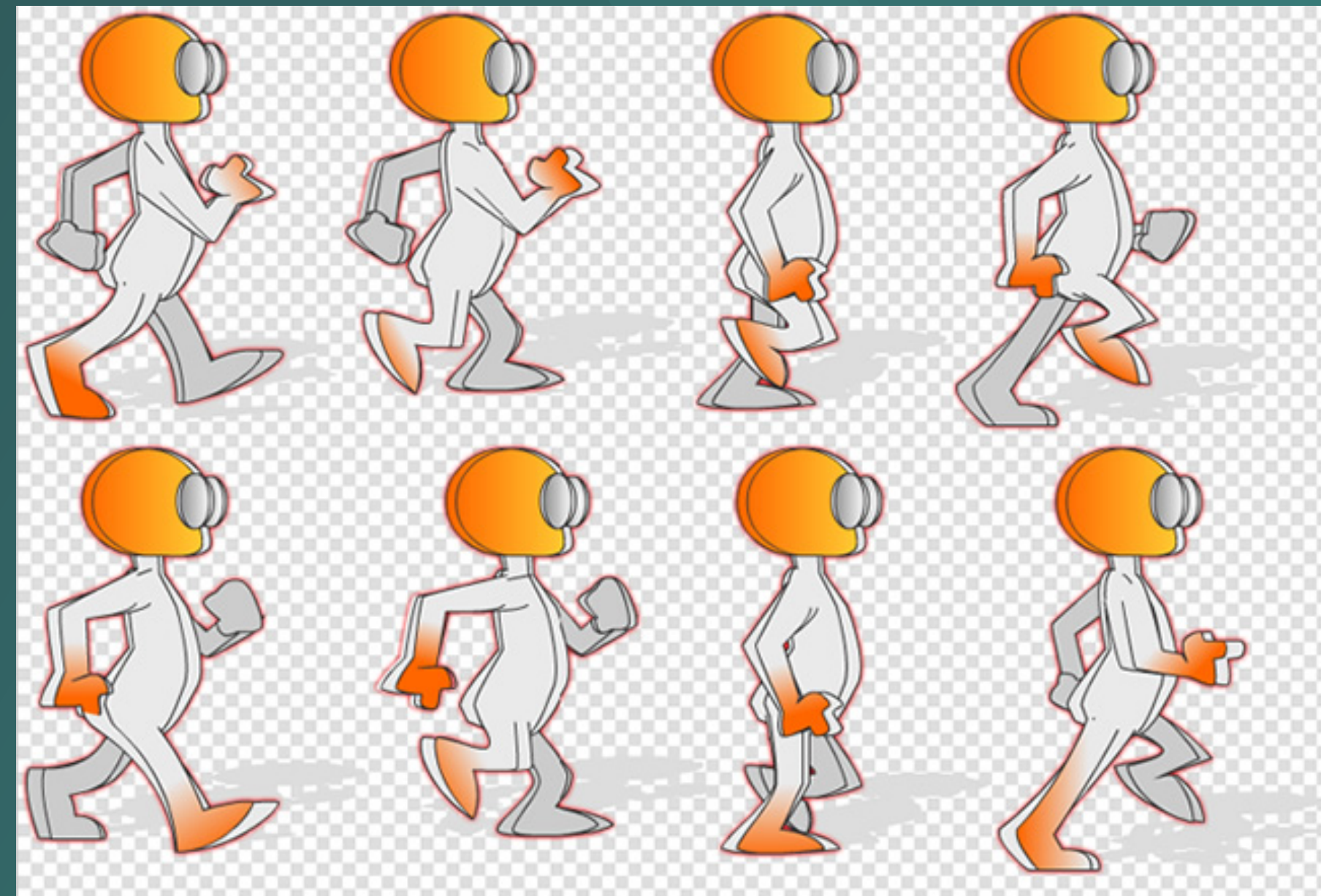
DEMO

Define: res/xml/animate_scale.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <!-- the pressed state; increase x and y size to 150% -->
  <item android:state_pressed="true">
    <set>
      <objectAnimator android:propertyName="scaleX"
        android:duration="@android:integer/config_shortAnimTime"
        android:valueTo="1.5"
        android:valueType="floatType"/>
      <objectAnimator android:propertyName="scaleY"
        android:duration="@android:integer/config_shortAnimTime"
        android:valueTo="1.5"
        android:valueType="floatType"/>
    </set>
  </item>
  <!-- the default state; pressed state, set x and y size to 100% -->
  <item android:state_pressed="false">
```

Animate bitmaps

- Used to animate a graphic such as:
 - An icon.
 - Illustration.
- Drawable animation API.
- Defined statically with a drawable resource or at runtime.



Using an AnimationDrawable



```
private lateinit var rocketAnimation: AnimationDrawable
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="true">
    <item android:drawable="@drawable/rocket_thrust1" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust2" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust3" android:duration="200" />
</animation-list>

val rocketImage = findViewById<ImageView>(R.id.rocket_image).apply {
    setBackgroundResource(R.drawable.rocket_thrust)
    rocketAnimation = background as AnimationDrawable
}

rocketImage.setOnClickListener({ rocketAnimation.start() })
}
```

<https://developer.android.com/guide/topics/graphics/drawable-animation>

Reveal or hide a view using animation

Create a crossfade animation

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/content"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <TextView style="?android:textAppearanceMedium"
            android:lineSpacingMultiplier="1.2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/lorem_ipsum"
            android:padding="16dp" />
    </ScrollView>

    <ProgressBar android:id="@+id/loading_spinner"
        style="?android:progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center" />

</FrameLayout>
```

<https://developer.android.com/training/animation/reveal-or-hide-view>

Reveal or hide a view using animation

Set up the crossfade animation

```
class CrossfadeActivity : Activity() {  
  
    private lateinit var mContentView: View  
    private lateinit var mLoadingView: View  
    private var mShortAnimationDuration: Int = 0  
    ...  
    override fun onCreate(savedInstanceState: Bundle?) {  
        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.visibility = View.GONE  
  
        // Retrieve and cache the system's default "short" animation time.  
        mShortAnimationDuration =  
            resources.getInteger(android.R.integer.config_shortAnimTime)  
    }  
    ...  
}
```

<https://developer.android.com/training/animation/reveal-or-hide-view>

Reveal or hide a view using animation

Crossfade the views

```
private fun crossfade() {  
    mContentView.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(mShortAnimationDuration.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.)  
    mLoadingView.animate()  
        .alpha(0f)
```

<https://developer.android.com/training/animation/reveal-or-hide-view>

Move a View with Animation

DEMO

Add curved motion

```
// arcTo() and PathInterpolator only available on API 21+
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
    val path = Path().apply {
        ObjectAnimator.ofFloat(view, "translationX", 100f).apply {
            arcTo(0f, 1000f, 1000f, 270f, -180f, true)
            duration = 2000
        }
        start()
    }
    val pathInterpolator = PathInterpolator(path)
}

<pathInterpolator xmlns:android="http://schemas.android.com/apk/res/android"
    val animation = ObjectAnimator.ofFloat(view, "translationX", 100f).apply {
        android:controlX1="0.4"
        interpolator = pathInterpolator
        android:controlY1="0"
        start()
        android:controlX2="1"
    }
    android:controlY2="1"/>
```

<https://developer.android.com/training/animation/reposition-view>

Animate Movement using Spring Physics

```
findViewById<View>(R.id.imageView).also { img ->
    SpringAnimation(img, DynamicAnimation.TRANSLATION_Y).apply {
        ....
        dependencies {
            // Setting the velocity with a constant speed
            implementation('com.soylentinc:spring-ktx:1.1.0-alpha03')
        }
        ..val velocity = vt.yVelocity
    }
}
val springAnim = findViewById<View>(R.id.imageView).let { img ->
    // Setting up a spring animation to animate the view's translationY property with the final
    // spring position at 0.
    SpringAnimation(img, DynamicAnimation.TRANSLATION_Y, 0f)
}
```


Stiffness



Figure 6: High stiffness



Figure 7: Medium stiffness



Figure 8: Low stiffness



Figure 9: Very low stiffness

Auto Animate Layout Updates

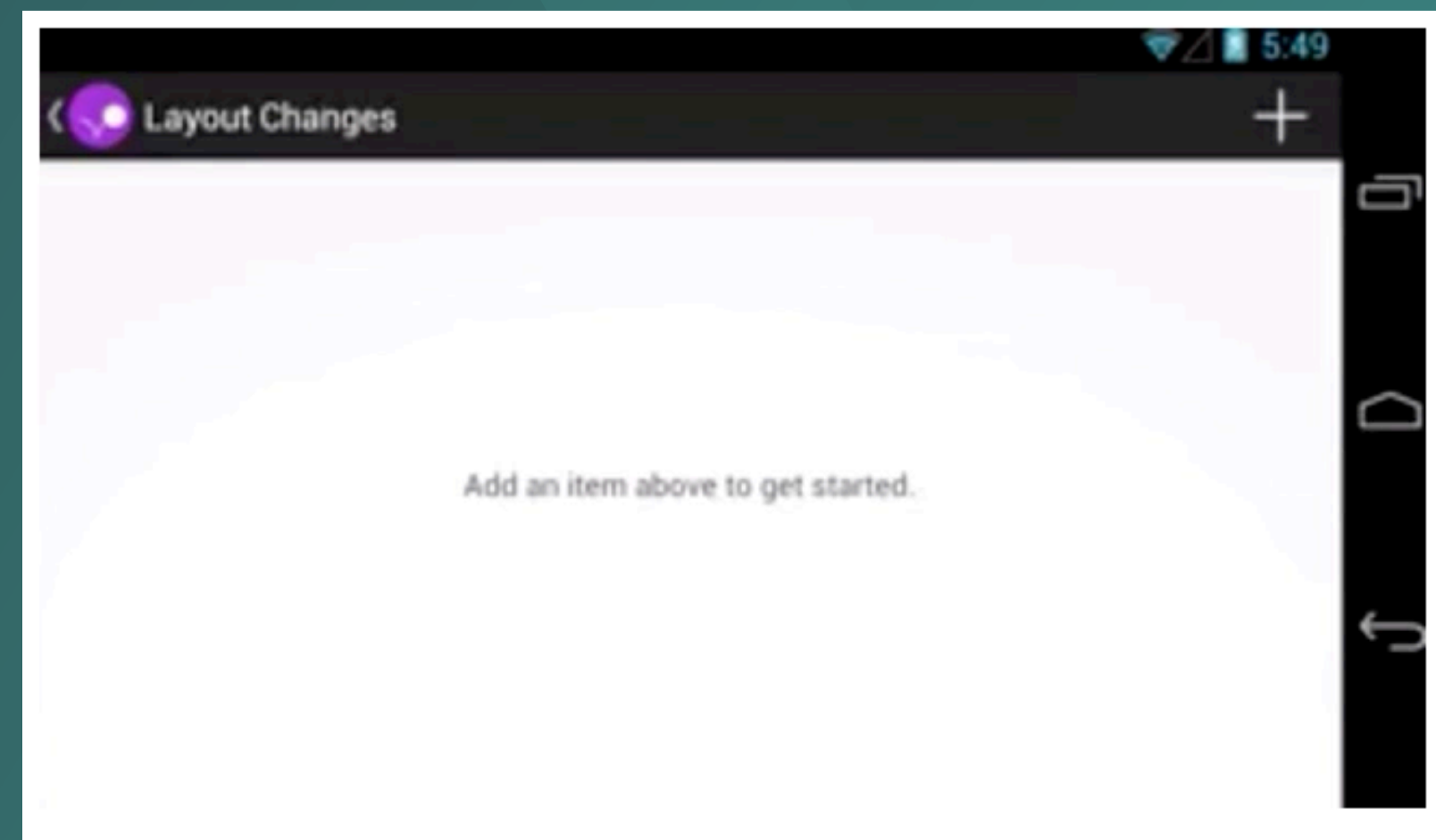
DEMO

Create the layout

```
<LinearLayout android:id="@+id/container"
    android:animateLayoutChanges="true"
    ...
/>
```

Add, update, or remove items from the layout

```
lateinit var mContainerView: ViewGroup
...
private fun addItem() {
    val newView: View = ...
    mContainerView.addView(newView, 0)
}
```



Animate Layout Changes Using Transitions

Define layouts for scenes

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/master_layout">
    <TextView
        android:id="@+id/title"
        ...
        android:text="Title"/>
    <FrameLayout
        android:id="@+id/scene_root">
        <include layout="@layout/a_scene" />
    </FrameLayout>
</LinearLayout>
res/layout/another_scene.xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/scene_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >
        <TextView
            <TextView
                android:id="@+id/text_view2"
                android:id="@+id/text_view1"
                android:text="Text Line 2" />
                android:text="Text Line 1" />
            <TextView
                android:id="@+id/text_view1"
                android:id="@+id/text_view2"
                android:text="Text Line 1" />
                android:text="Text Line 2" />
        </RelativeLayout>
    </RelativeLayout>
```


Create the Scene

DEMO

Generate scenes from layouts

```
val mSceneRoot: ViewGroup = findViewById(R.id.scene_root)
val mAScene: Scene = Scene.getSceneForLayout(mSceneRoot, R.layout.a_scene, this)
val mAnotherScene: Scene = Scene.getSceneForLayout(mSceneRoot,
    R.layout.another_scene, this)
```

Create a scene in your code

```
val mSceneRoot = mSomeLayoutElement as ViewGroup
val mViewHierarchy = someOtherLayoutElement as ViewGroup
val mScene: Scene = Scene(mSceneRoot, mViewHierarchy)
```

Apply a transition

```
var mFadeTransition: Transition =
    TransitionInflater.from(this)
        .inflateTransition(R.transition.fade_transition)

var mFadeTransition: Transition = Fade()

TransitionManager.go(mEndingScene, mFadeTransition)
```

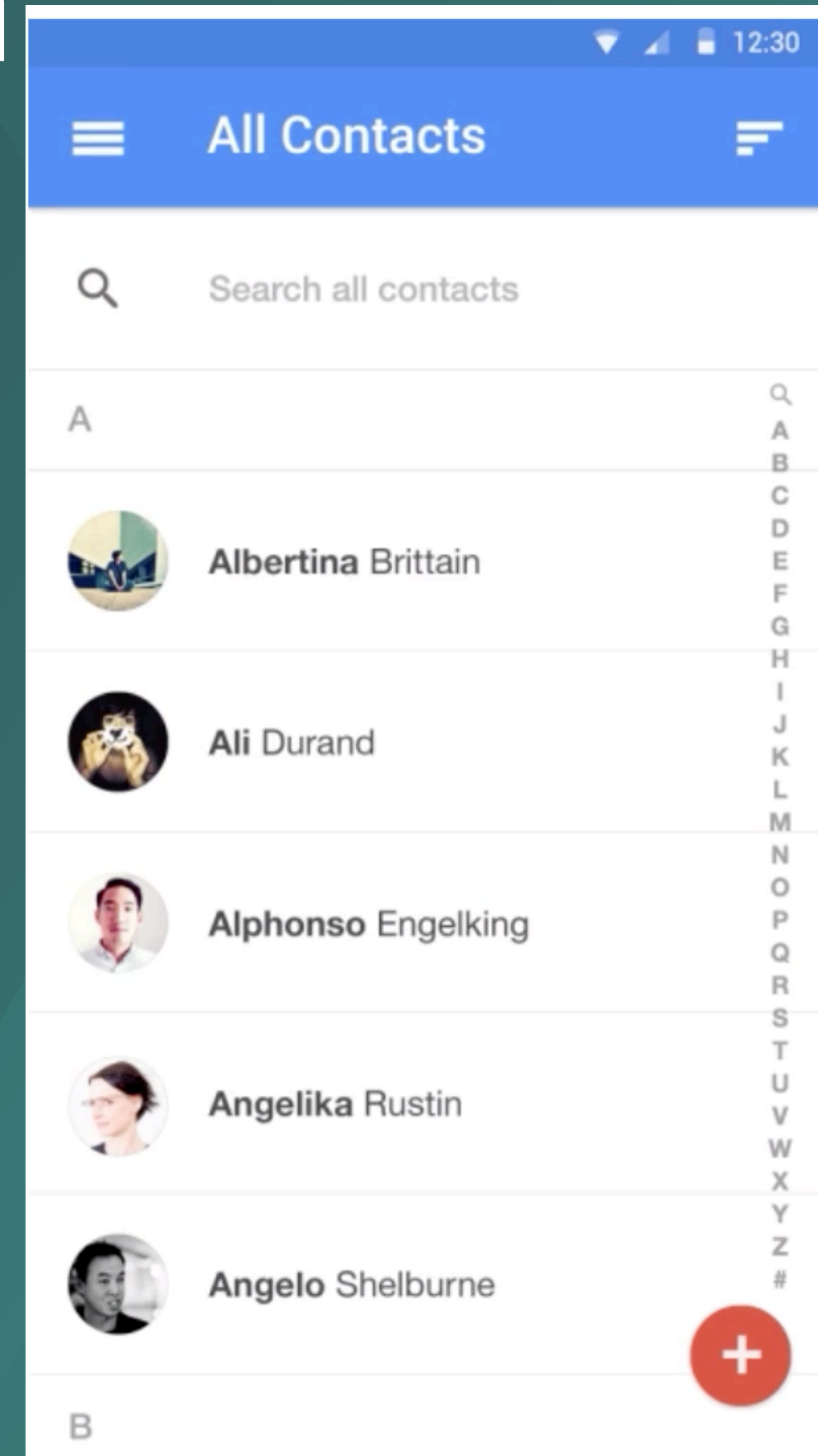

Start an Activity using an Animation

DEMO

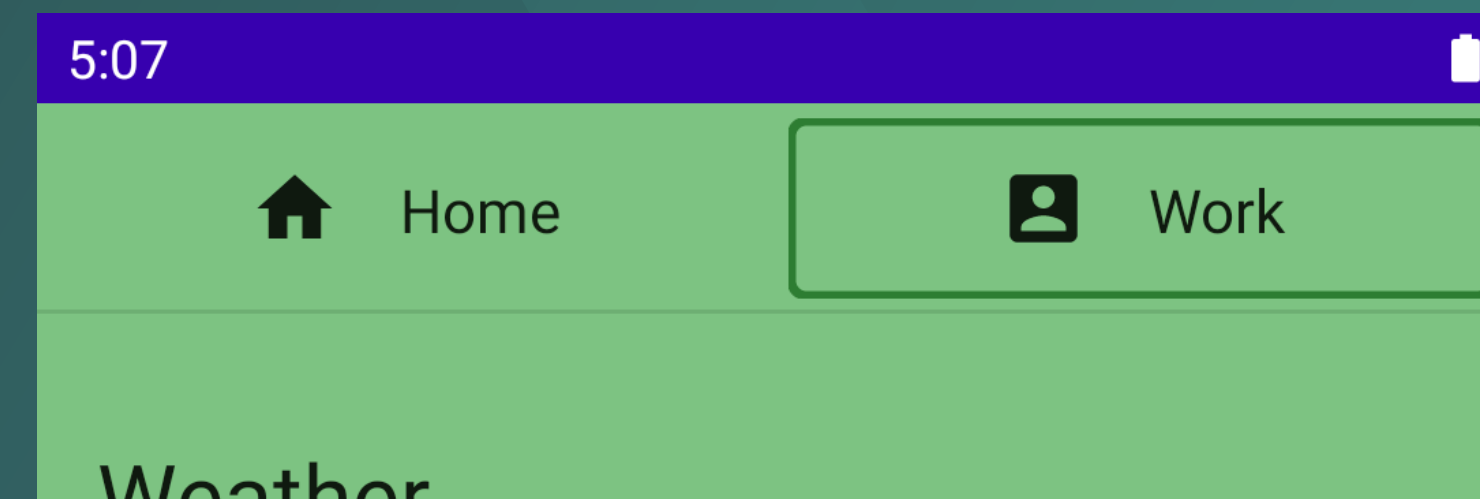
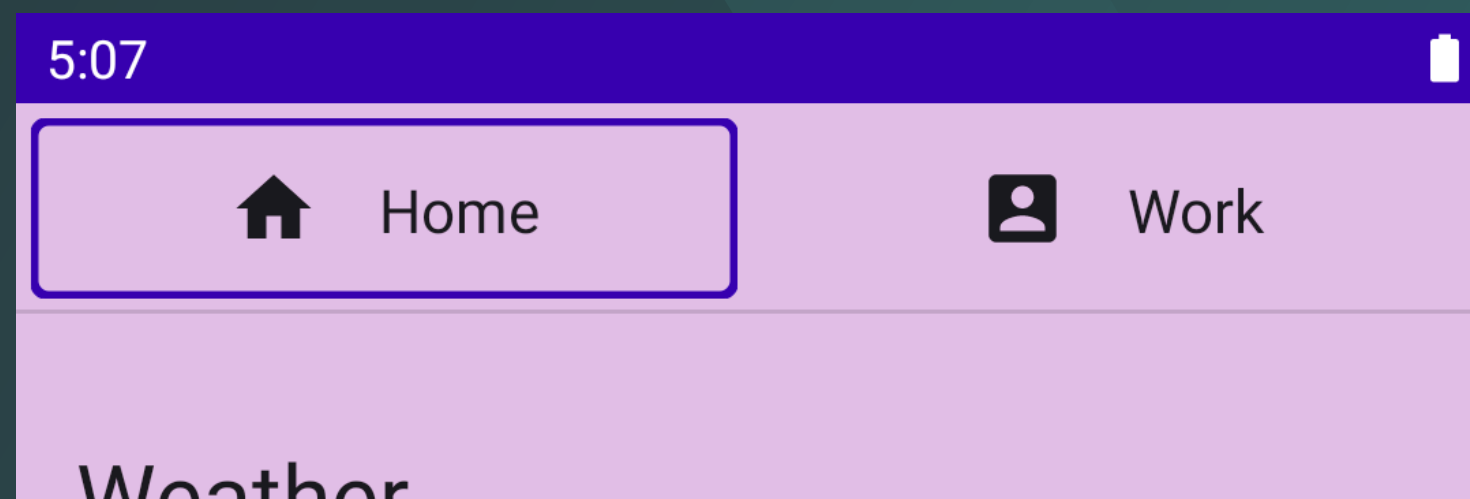
```
// get the element that receives the click event
val imgContainerView =
    findViewById<View>(R.id.img_container)

// get the common element for the
// transition in this activity from the
// Android framework to avoid a name clash
val androidRobotView =
    findViewById<View>(R.id.image_small)
import android.util.Pair as UtilPair

// define a click listener
val options =
    imgContainerView.setOnClickListener({
        ActivityOptions.makeSceneTransitionAnimation(
            val intent = Intent(this, Activity2::class.java)
            // create the transition animation
            UtilPair.create(view1, "agreedName1"),
            // - the images in the layouts
            UtilPair.create(view2, "agreedName2")
            // or both activities are defined
        // with android:transitionName="robot"
        val options = ActivityOptions
            .makeSceneTransitionAnimation(
                this, androidRobotView, "robot")
        // start the new activity
        startActivity(intent, options.toBundle())
    })
```

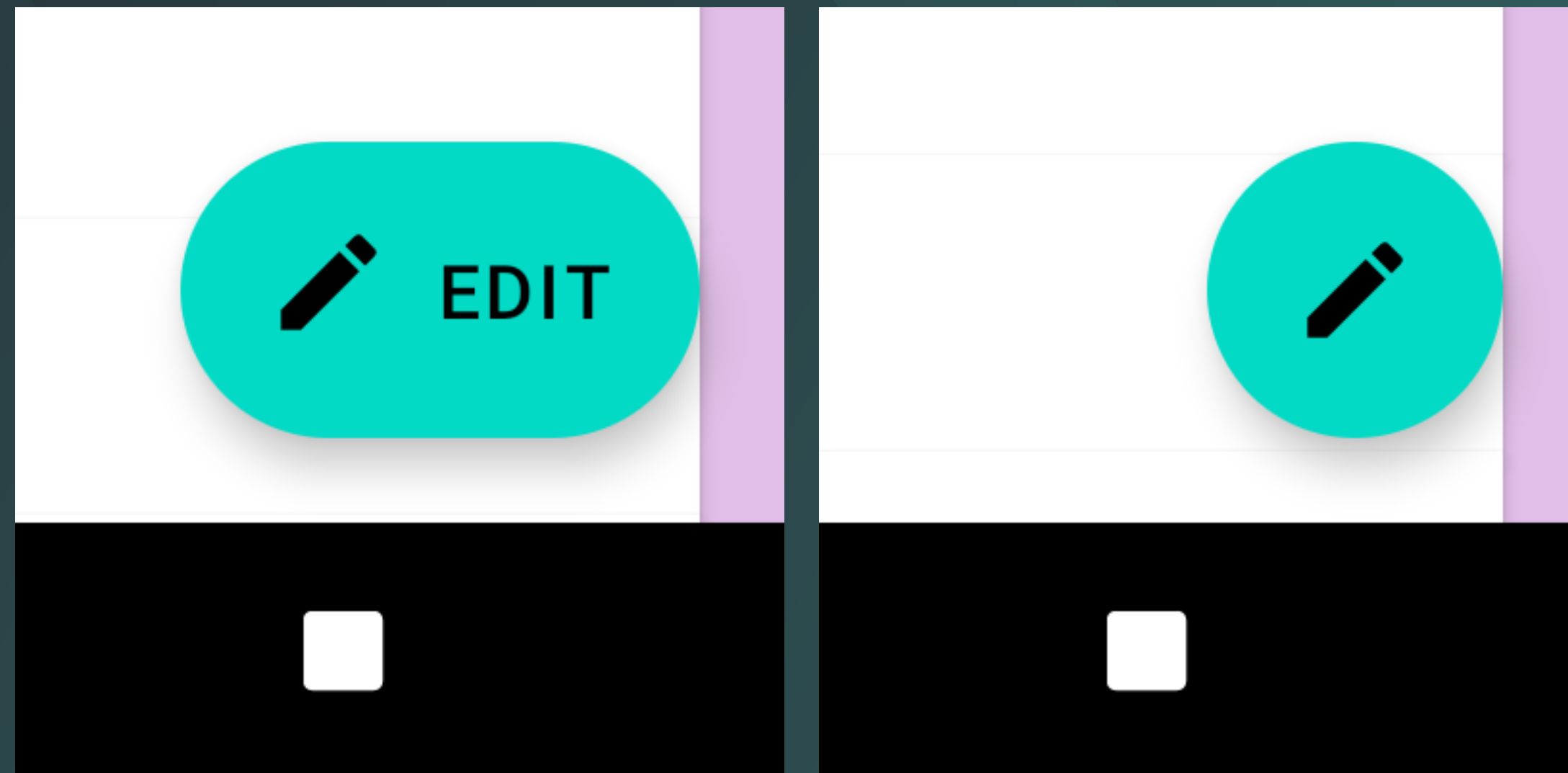


Animating a simple value change



```
val backgroundColor by animateColorState(TabPage.Home) TabPage.Home) Purple300 else Green300)
```

Animating visibility



```
AnimatedVisibility(extended) {  
    Text(  
        text = stringResource(R.string.edit),  
        modifier = Modifier  
            .padding(start = 8.dp, top = 3.dp)  
    )  
}
```


Animating visibility



Topics

- i** 2 new packages arrived
- i** DIY project recommendation
- i** Festival next month
- i** New flower seeds available

Topics

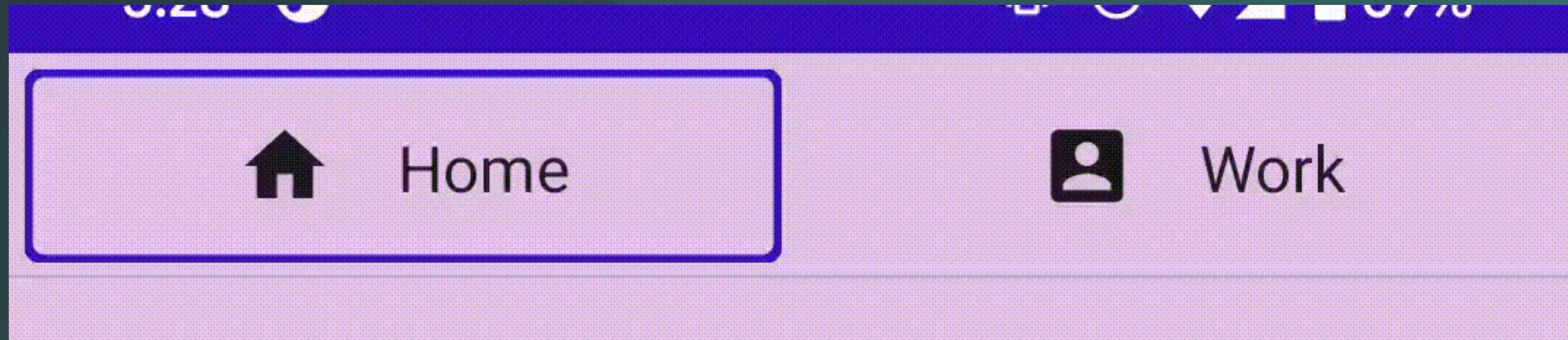
- i** 2 new packages arrived

- i** DIY project recommendation

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

```
Column(  
  modifier = Modifier  
    .fillMaxWidth()  
    .padding(16.dp)  
) {  
  .animateContentSize()  
} // ... the title and the body  
} // ... the title and the body  
}
```

Multiple value animation

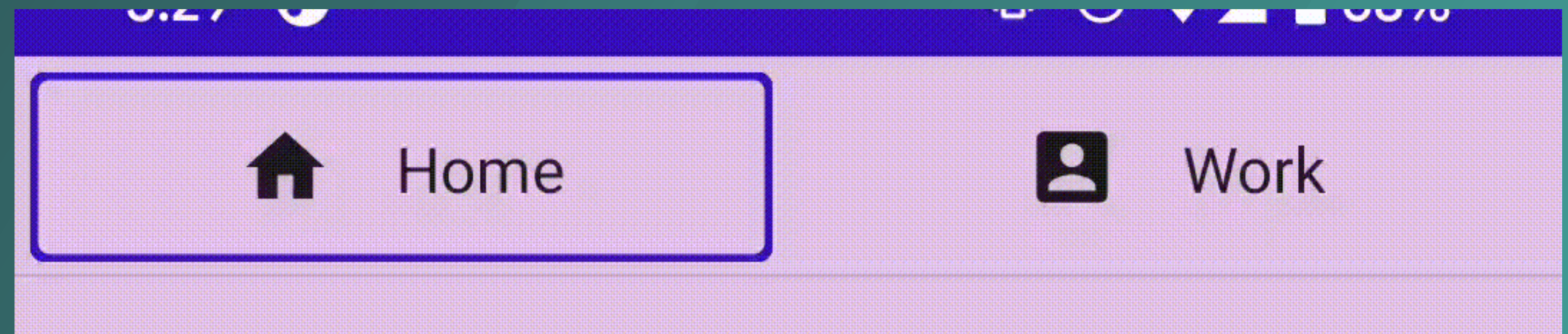


```
val transition = updateTransition(  
    tabPage,  
    label = "Tab indicator"  
)  
val indicatorLeft by transition.animateDp { page -> tabPositions[page.ordinal].left }  
val indicatorRight by transition.animateDp { page -> tabPositions[page.ordinal].right }  
val color by transition.animateColor { page ->  
    if (page == TabPage.Home) Purple700 else Green800  
}
```


Multiple value animation



```
val indicatorLeft by transition.animateDp(
    transitionSpec = {
        if (TabPage.Home isTransitioningTo TabPage.Work) {
            // Indicator moves to the right.
            // The left edge moves slower than the right edge.
            spring(stiffness = Spring.StiffnessVeryLow)
        } else {
            // Indicator moves to the left.
            // The left edge moves faster than the right edge.
            spring(stiffness = Spring.StiffnessMedium)
        }
    },
    label = "Indicator left"
) { page -> tabPositions[page.ordinal].left}
```

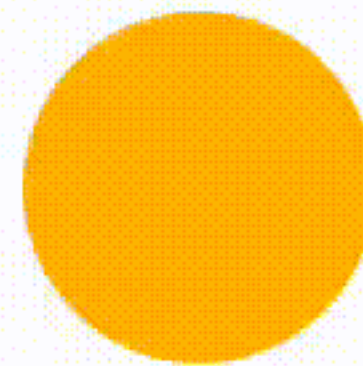


Repeated animation



```
val infiniteTransition = rememberInfiniteTransition()
val alpha by infiniteTransition.animateFloat(
    initialValue = 0f,
    targetValue = 1f,
    animationSpec = infiniteRepeatable(
        animation = keyframes {
            durationMillis = 1000
            0.7f at 500
        },
        repeatMode = RepeatMode.Reverse
    )
)
```

Weather



18 °C



Lecture outcomes

- Animate bitmaps.
- Animate UI visibility and motion.
- Physics-based motion.
- Animate layout changes.
- Animate between activities.

