INTERMEDIATE

iOS ANIMATIONS

HANDS-ON CHALLENGES

Intermediate iOS Animations

Catie & Jessy Catterwaul

Notice of Rights

Notice of Liability

Trademarks

Table of Contents: Overview

Table of Contents: Extended

Challenge 2: Beginning Property Animators

By Catie & Jessy Catterwaul

Constraint Animations with Property Animators

In the Beginning iOS Animations course, you started out by learning how to animate constraints. In this challenge, you'll combine everything you know about animating constraints and using property animators!

Layout constraint animations with `UIViewPropertyAnimator` are very similar to how you create them with `UIView.animate(withDuration:...)`. The trick was to update a constraint, and then call `layoutIfNeeded()` from within an animations block.

Let's try the same with `UIViewPropertyAnimator`.

Open **AnimatorFactory.swift** and add a new factory method:

```
@discardableResult
static func animateConstraint(
  view: UIView,
  constraint: NSLayoutConstraint,
  by: CGFloat
) -> UIViewPropertyAnimator {

}
```

In this method, you are going to animate a change to a constraint's constant and then call `layoutIfNeeded()` on the provided view.

What is this animation going to look like? It really depends on what kind of constraint you provide to the method. If you give it a trailing space constraint, it will move the view horizontally. If you provide it with a height constraint, the view will scale up or down.

Inside the method, create an animator:

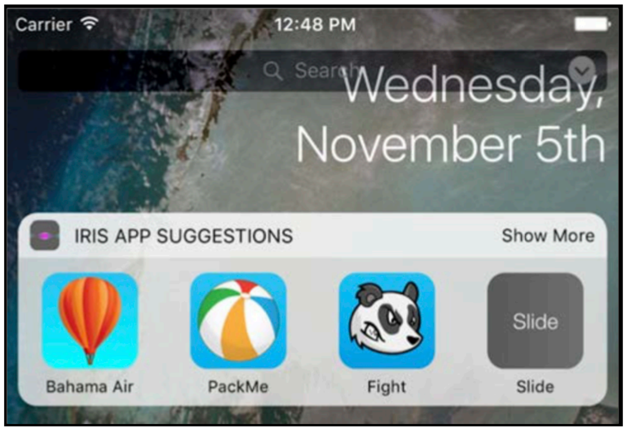
```
let spring = UISpringTimingParameters(dampingRatio: 0.2)
let animator = UIViewPropertyAnimator(duration: 2.0,
  timingParameters: spring)
animator.addAnimations {
  constraint.constant += by
  view.layoutIfNeeded()
}
return animator
```

Here you've created a basic spring animation! You create a spring using only `dampingRatio`, use the convenience initializer that takes `timingParameters`, then you simply change the constraint and trigger an Auto Layout pass.

Now switch to **LockScreenViewController.swift** and add to `viewWillAppear(_:)`:

```
dateTopConstraint.constant -= 100
view.layoutIfNeeded()
```

This will move the date label up by 100 points like so:

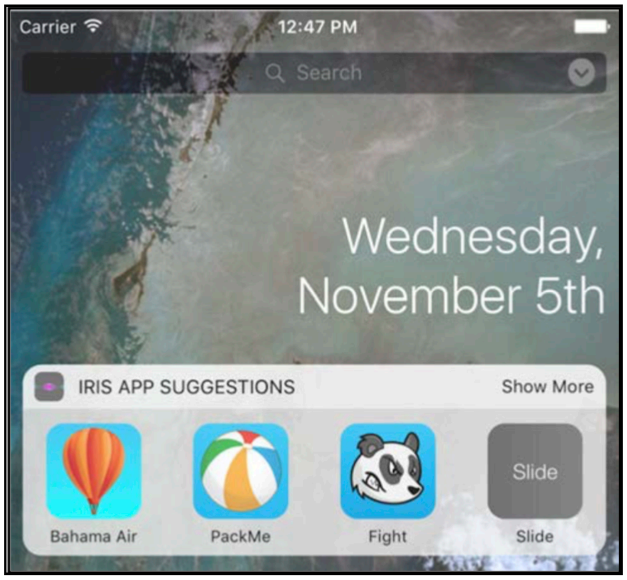


Next, trigger an animation to move the label, and all other views attached to it, down to its original location.

Append the following to `viewDidAppear(_:)`:

```
AnimatorFactory.animateConstraint(
  view: view,
  constraint: dateTopConstraint,
  by: 100
).startAnimation()
```

Run the app and check out the resulting animation! Since you are moving all of the views down and scaling up your table view, this results in a rather complex yet smooth transition:



The animation looks a bit excessive though. The first couple of times, it looks nice, but if your users see this bouncy transition multiple times every day they are certainly going to hate your app.

This is a nice reminder that UI spring animations should be all about moderation.

Switch back to **AnimatorFactory.swift** and change the parameters of your spring animation. Set `dampingRatio` to 0.55, and `duration` to 1.0. This should make the animation more subtle, yet still playful.