iOS Animation with Swift

Hands-On Challenges

iOS Animation with Swift Hands-On Challenges

Copyright © 2014 Razeware LLC.

All rights reserved. No part of this book or corresponding materials (such as text, images, or source code) may be reproduced or distributed by any means without prior written per- mission of the copyright owner.

This book and all corresponding materials (such as source code) are provided on an "as is" basis, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose, and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in action of contract, tort or otherwise, arising from, out of or in connec- tion with the software or the use or other dealings in the software.

All trademarks and registered trademarks appearing in this book are the property of their respective owners.

Challenge H: Layer Keyframe animations

In this challenge you'll get to exercise keyframe animations and create one on your own. The challenge is pretty similar to what you did back in the `UIView` keyframe animations tutorial challenge so I expect you to do just fine.

Part 1: Simple keyframe movement

Scroll to `viewWillAppear(animated:)` and remove the two lines that set the initial login button position:

```
loginButton.layer.position.y += 100.0
loginButton.layer.opacity = 0.0
```

Then move to `viewDidAppear(animated:)` and remove all the code that created a group animation on the login button:

```
let flyUpAndFadeIn = CAAnimationGroup()
flyUpAndFadeIn.beginTime = CACurrentMediaTime() + 0.5
flyUpAndFadeIn.duration = 0.5
flyUpAndFadeIn.delegate = self
flyUpAndFadeIn.setValue("loginButton", forKey: "name")

let flyUp = CABasicAnimation(keyPath: "position.y")
flyUp.toValue = loginButton.layer.position.y - 100

let fadeIn = CABasicAnimation(keyPath: "opacity")
fadeIn.toValue = 1.0

flyUpAndFadeIn.animations = [flyUp, fadeIn]

loginButton.layer.addAnimation(flyUpAndFadeIn, forKey: nil)
```

Don't mourn that piece of code though, you are going to create your own and way more awesome keyframe-animation. On the spot where the group-animation code used to be place this new one:

```
let loginButtonAnimation = CAKeyframeAnimation(keyPath:
"position")
loginButtonAnimation.duration = 0.6
loginButtonAnimation.values = [
  NSValue(CGPoint: CGPoint(x: -view.frame.size.width/2, y:
loginButton.layer.position.y+100)),
  NSValue(CGPoint: CGPoint(x: view.frame.size.width/2, y:
loginButton.layer.position.y+100)),
  NSValue(CGPoint: CGPoint(x: view.frame.size.width/2, y:
loginButton.layer.position.y))
]
loginButtonAnimation.keyTimes = [0.0, 0.5, 1.0]
loginButtonAnimation.additive = false
loginButton.layer.addAnimation(loginButtonAnimation, forKey: nil)
```

Notice how you didn't need to set the initial position in `viewWillAppear`? This is because you can set the initial position as the `value` of position at `keyTime 0.0`.

You specify the 3 absolute positions on screen of the button along the animation and set additive to false since CoreAnimation is not to add the values, but they are absolute values you want to have at the given key times.

Part 2: Cloud layer animations

In this challenge you will re-create the cloud movement in the background of the by using layer animations.

Find `animateCloud(cloud:)` and delete it from your view controller class. You will rewrite the whole method with CoreAnimation. First let's set the animation constants – add the empty method body:

```
func animateCloud(cloud: UIImageView) {
  let cloudSpeed = 20.0 / Double(view.layer.frame.size.width)
  let duration: NSTimeInterval =
Double(view.layer.frame.size.width - cloud.layer.frame.origin.x) *
cloudSpeed

}
```

This is very similar code to what you had before but it uses the layer's frame to calculate the cloud speed.

Next – create the position animation to move the clouds sideways across the screen. Add to `animateCloud(cloud:)`:

```
let cloudMove = CABasicAnimation(keyPath: "position.x")
cloudMove.duration = duration
cloudMove.toValue = self.view.bounds.size.width
cloudMove.delegate = self
cloudMove.setValue("cloud", forKey: "name")
cloudMove.setValue(cloud, forKey: "view")

cloud.layer.addAnimation(cloudMove, forKey: nil)
```

You assign the cloud view to the animation and set a name for the animation object because you will want to adjust the cloud layer whenever the animation completes running.

In `animationDidStop` add a new if to handle the new animation:

```
if name == "cloud" {
  let cloud: UIImageView = anim.valueForKey("view") as UIImageView
  cloud.frame.origin.x = -self.cloud1.frame.size.width
  delay(seconds: 0.1, {
    self.animateCloud(cloud);
  })
}
```

This code resets the position of the cloud to outside of the screen bounds and after a short delay restarts the cloud animation.

If you'd like to experiment further you can try changing the animation speed or make it use a different timing function.