Beginning iOS Animation

Hands-on challenges

Beginning iOS Animation Hands-On Challenges

Copyright © 2015 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.

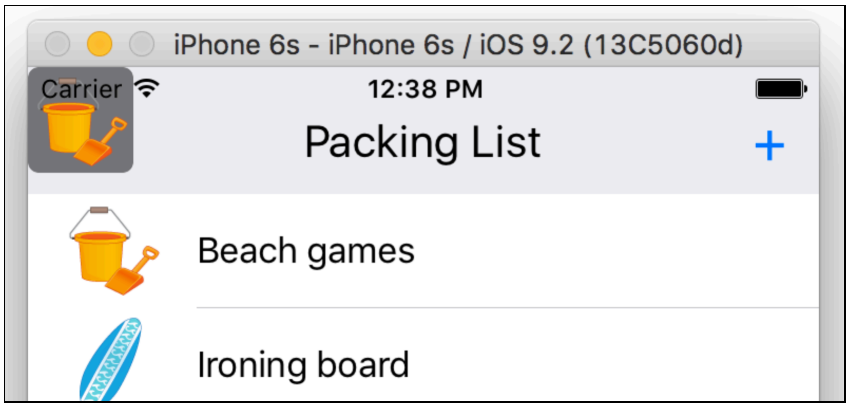
Challenge A: Creating constraints

In this video you learned how to create animations by finding and replacing constraints with new ones. In the process you also learned how to create a new constraint object from code and add it to the layout hierarchy.

To exercise everything you've learned so far about animating constraints you are going to finish the core functionality of the packing list project by adding an action method to display a preview of any item the user selects in the list.

This will require you to create all the constraints to position and size the preview and then create animations to show and hide the preview.

To start open **ViewController.swift** and find the `showItem(_:)` method. This method is already connected to the event produced by the user tapping on a table row - in fact you can tap on an item yourself and see the item preview show up in the top left corner:



The image view shows on that spot because there aren't any constraints to position it elsewhere. This is something you are going to fix right now :]

Add this code at the bottom of `showItem(_:)` to create a horizontal alignment constraint:

```
let conX =
imageView.centerXAnchor.constraintEqualToAnchor(view.centerXAnchor
)
```

This will center the image view in the view controller's view. Next add a constraint to vertically position the image view in its parent view:

```
let conBottom =
imageView.bottomAnchor.constraintEqualToAnchor(view.bottomAnchor,
constant: imageView.frame.height)
```

This will "anchor" the image view to the bottom of the view controller but leave a bit of margin - specifically a margin as big as the image view's own height. Next set the image's width:

```
let conWidth =
imageView.widthAnchor.constraintEqualToAnchor(view.widthAnchor,
multiplier: 0.33, constant: -50.0)
```

This makes the image view a third of the view controller's width minus 50 points. Why the 50 points subtraction? You will see in a minute.

Finally add the last constraint to fix the image view's size ratio:

```
let conHeight = imageView.heightAnchor.constraintEqualToAnchor(
imageView.widthAnchor)
```

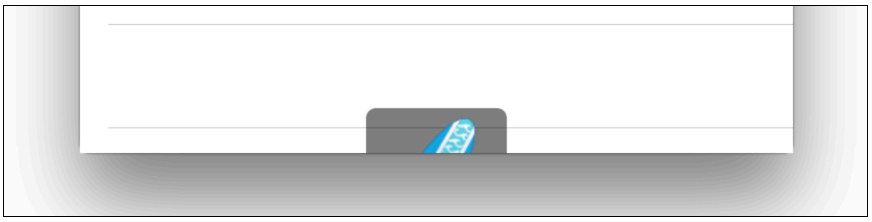
Now you have all four constraints needed to position and size the item preview. The only thing left is to activate them.

You will use the method to batch activate constraints - add:

```
NSLayoutConstraint.activateConstraints([conX, conBottom, conWidth,
conHeight])
view.layoutIfNeeded()
```

This adds the constraints and forces Auto Layout to update the app's layout.

Run the app now to see your new constraints at work:



But why is the preview positioned so awkwardly? Well this is just the starting position for the animation that will display the item!

Now you can add also the animation code:

```
conBottom.constant = -imageView.frame.size.height/2
conWidth.constant = 0.0
UIView.animateWithDuration(0.33, delay: 0.0,
  options: [.CurveEaseOut], animations: {
```

```
      self.view.layoutIfNeeded()
    }, completion: nil)
```

This will reposition the preview vertically by changing the `conBottom` constraint and will reset the constant property on the `conWidth` constraint. The animation will make the preview move upward and grow a bit in size.

Run the app to check the effect:

After a while you'd want to hide the preview automatically. Add one last animation to the method:

```
UIView.animateWithDuration(0.67, delay: 2.0,
  options: [], animations: {
    conBottom.constant = imageView.frame.size.height
    conWidth.constant = -50.0
    self.view.layoutIfNeeded()
}, completion: {_ in

})
```

This code gives the animation `2` seconds of delay and animates the preview out of the screen the same way it appeared but in reverse. Once the preview is out of the visible screen area you do not need the image view anymore so you can remove it from the view hierarchy.

Add inside the `completion` closure:

```
imageView.removeFromSuperview()
```

That's all for this challenge! In the next video you are going to learn about springs - boing yoink!