Simple Animation of UIView

This tutorial is on simple animation of UIView using the class methods:

```
animateWithDuration(duration: NSTimeInterval,
                    animations: () -> Void)

animateWithDuration(duration: NSTimeInterval,
                    animations: () -> Void),
                    completion: ((Bool) -> Void)?)
```

Both methods have the parameter duration of type NSTimeInterval. The value assigned to this parameter will determine how long the animation will take from start to finish. The parameter animations is a closure, where we place all the changes to animatable properties of UIView. The second method has an additional parameter: completion, which is also a closure. We place all the clean up code here.

In this tutorial, we will animate the position, background color, and size of a UIView placed on the interface. A setting of a UISwitch will determine which method is to be used. Pressing a button will start the animation.

We start by creating a single view application in Xcode. On the interface we place a UIView control and resize it to be of dimensions 50 x 50 giving it a background color of red. We also place a UISwitch and UIButton. The controls are positioned on the interface using appropriate constraints. We need to create outlet for the UIView:

```
@IBOutlet weak var myView: UIView!
```

We need to define the follow variables:

```
var completionOption = false
var originalCenter:CGPoint!
var originalBounds: CGRect!
var originalColor:UIColor!
```

The Bool variable completionOption is true if the switch is on and false if the switch is off. We initialize the switch in off position. So this variable is initialized to be false. The variable originalCenter is of a CGPoint type to store the initial value of the center of myView. Similarly, the variable originalBounds stores the initial value for the view's bounds and the originalColor stores the initial background color of the view. The best place to put these initialization in the viewDidAppear function:

```
    override func  viewDidAppear(animated: Bool) {
        originalCenter = myView.center
        originalBounds = myView.bounds
        originalColor = myView.backgroundColor
        myView.bounds = CGRectMake(0, 0, 50, 50)
    }
```

The UISwitch is hooked to the following method:

```
@IBAction func changeCompletionOption(sender: AnyObject) {
    let option = sender as UISwitch
    if option.on {
        completionOption = true
    }
    else{
        completionOption = false
    }
    //Restore the center and background color to initial values
    myView.center = originalCenter
    myView.bounds = originalBounds
    myView.backgroundColor = originalColor

}
```

If the switch is on, the completionOption is set true. Otherwise it is set false. When the switch state is changed, it is necessary to restore myView to its initial state.

The following function contains the animation block using the first method without a completion block:

```
func simplAnimation(){

    UIView.animateWithDuration(2.0,  animations: {
        var center = self.myView.center
        //Shift view right without animation
        UIView.performWithoutAnimation({
            center.x += 200
            self.myView.center = center
        })
        center.y += 600
        self.myView.center = center
        self.myView.bounds = CGRectMake(0, 0, 100, 100)
        self.myView.backgroundColor = UIColor.blueColor()
    })
}
```

Inside the animation block, we have called the method performWithoutAnimation, which has a closure as a parameter. The code inside this method moves myView to the right by 200 points without animation. Instead of doing this, we could have placed the code in this method outside the animation block. Purpose of putting the change inside the animation block is to show how we can include code inside the animation that does not animate. When we call this function, the view will abruptly move to the right by 200 points and then animate down by 600 points with background color changing to blue.

The following function is called when we use a completion block to return the view to its initial state at the end of the animation:

```
func animateWithReturn(){

    UIView.animateWithDuration(5.0,animations: {
        var center = self.myView.center
        UIView.performWithoutAnimation({
            center.x += 200
            self.myView.center = center
        })
        center.y += 600
        self.myView.center = center
        self.myView.bounds = CGRectMake(0, 0, 100, 100)
        self.myView.backgroundColor = UIColor.greenColor()
        }, completion:{
            finished in
            self.myView.center = self.originalCenter
            self.myView.bounds = self.originalBounds
            self.myView.backgroundColor = self.originalColor

    })

}
```

At the end of the animation, the view will now be returned to its original position and background color of red.

We hook up the button (with title Animate) to the following method:

```
@IBAction func buttonPressed(sender: AnyObject) {

    if completionOption {
        animateWithReturn()
        }
    else{

        simplAnimation()
        }
}
```

When the button is pressed, the appropriate function will be called depending on the state of the switch.