Swift

ARC

- weak var tenant: Person?
- unowned let customer: Customer (не может быть nil)
- unwoned(unsafe) аналог __unsafe_unretained, в отличии от unowned(safe) не создает исключения, если объект удален

is

```
func doSomething(someObject: AnyObject) {
        if someObject is NSString {
            println("is string")
        }
    }
```

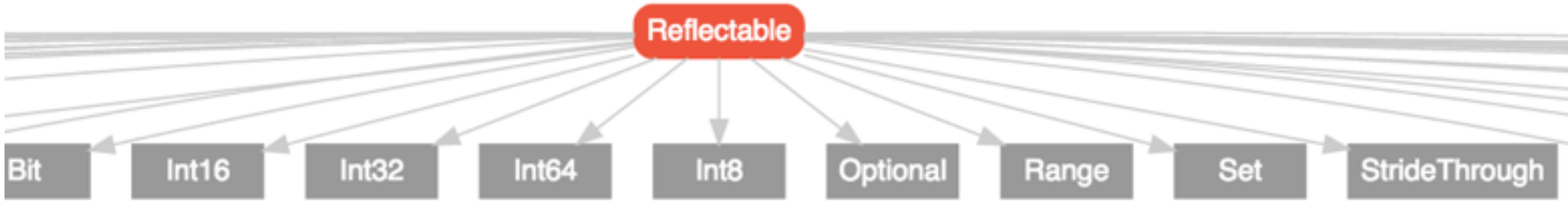
dynamicType

```
class SomeBaseClass {
    class func printClassName() {
        println("SomeBaseClass")
    }
}
class SomeSubClass: SomeBaseClass {
    override class func printClassName() {
        println("SomeSubClass")
    }
}
let someInstance: SomeBaseClass = SomeSubClass()
someInstance.dynamicType.printClassName()
```

Reflectable

```
/// Customizes the result of `reflect(x)`, where `x` is a conforming
/// type.
protocol Reflectable {

    /// Returns a mirror that reflects `self`.
    func getMirror() -> MirrorType
}
```

http://swiftdoc.org/protocol/Reflectable/hierarchy/



MirrorType

```
/// The type returned by `reflect(x)`; supplies an API for runtime
/// reflection on `x`
protocol MirrorType {

    /// The instance being reflected
    var value: Any { get }

    /// Identical to `value.dynamicType`
    var valueType: Any.Type { get }

    /// A unique identifier for `value` if it is a class instance; `nil`
    /// otherwise.
    var objectIdentifier: ObjectIdentifier? { get }

    /// The count of `value`\ 's logical children
    var count: Int { get }
    subscript (i: Int) -> (String, MirrorType) { get }

    /// A string description of `value`.
    var summary: String { get }

    /// A rich representation of `value` for an IDE, or `nil` if none is supplied.
    var quickLookObject: QuickLookObject? { get }

    /// How `value` should be presented in an IDE.
    var disposition: MirrorDisposition { get }
}
```

MirrorDisposition

```
/// How children of this value should be presented in the IDE.
enum MirrorDisposition {
    case Struct
    case Class
    case Enum
    case Tuple
    case Aggregate
    case IndexContainer
    case KeyContainer
    case MembershipContainer
    case Container
    case Optional
    case ObjCObject
}
```

QuickLookObject

```
enum QuickLookObject {
    case Text(String)
    case Int(Int64)
    case UInt(UInt64)
    case Float(Float32)
    case Double(Float64)
    case Image(Any)
    case Sound(Any)
    case Color(Any)
    case BezierPath(Any)
    case AttributedString(Any)
    case Rectangle(Float64, Float64, Float64, Float64)
    case Point(Float64, Float64)
    case Size(Float64, Float64)
    case Logical(Bool)
    case Range(UInt64, UInt64)
    case View(Any)
    case Sprite(Any)
    case URL(String)
}
```

reflect

```
class SomeClass {
    let stringProperty = "Hello"
    let intProperty = 41
}
```

```
let someObject = SomeClass()
println("\(reflect(someObject)[0])")
```

(stringProperty, Swift._LeafMirror<Swift.String>)

Interface Builder

Designable

- обновляет View в IB

Designable

- IB_DESIGNABLE @interface MyDrawFrameRectViewClass : UIView

- @IBDesignable class MyCustomView: UIView {}

Designable

- перед обновлением view вызывается prepareForInterfaceBuilder

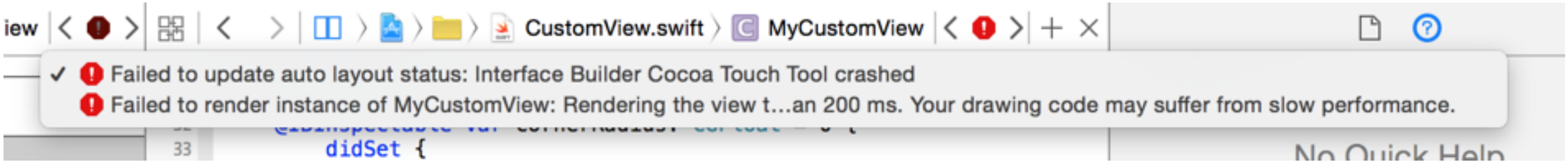
Designable

	APP running	`Debug Selected Views` (`Live Rendering`)	Function
path	`initWithCoder:` ⋮ ⬇ `drawRect:`	`initWithFrame:` ⋮ ⬇ `prepareForInterfaceBuilder` ⋮ ⬇ `drawRect:`	init method Live Render 1 Draw

Designable

IBDesignable не работает если есть только init(coder)

```
required init(coder aDecoder: NSCoder) {
    super.init(coder: aDecoder)
}
```



рабочий вариант

```
override init(frame: CGRect) {
    super.init(frame: frame)
}

required init(coder aDecoder: NSCoder) {
    super.init(coder: aDecoder)
}
```

Designable

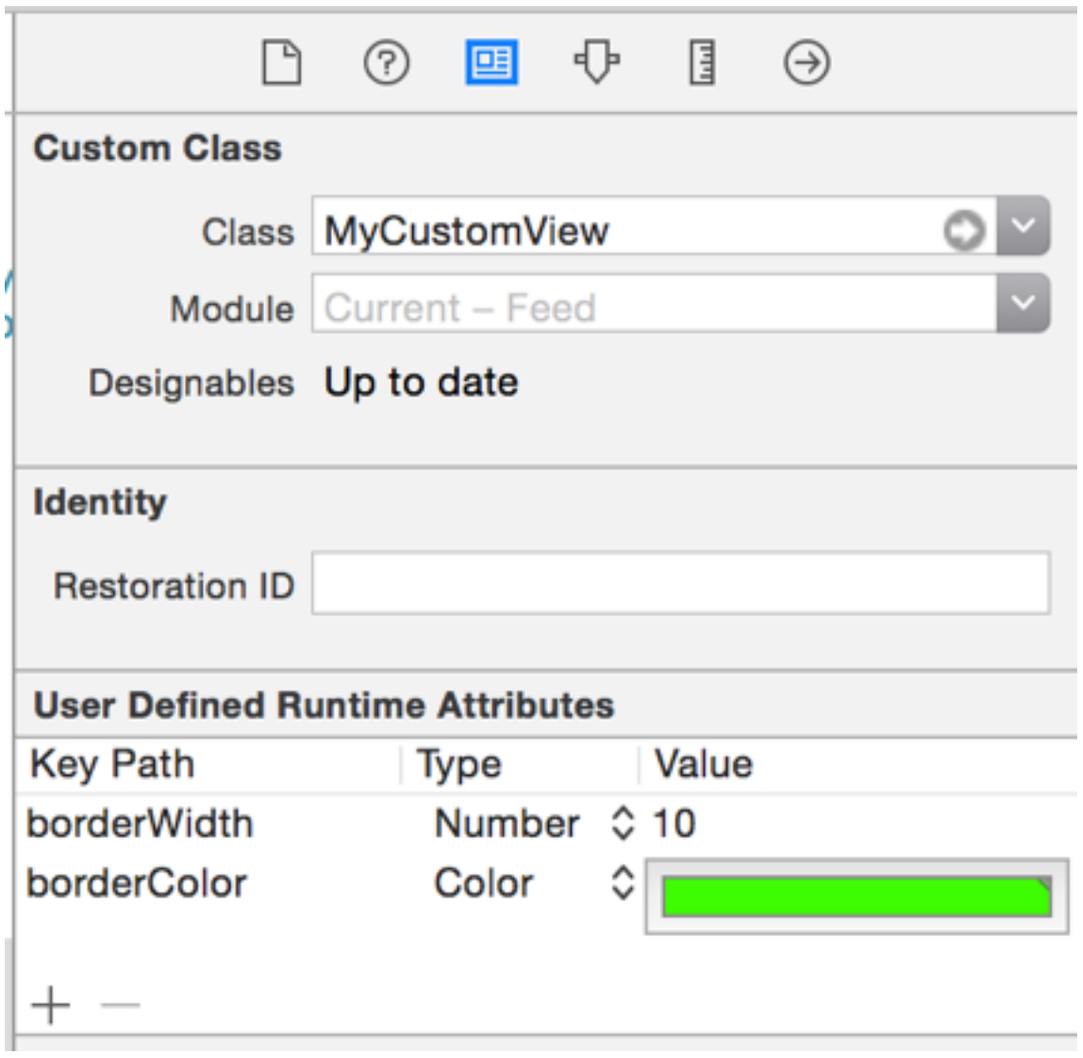
```
@IBDesignable
class MyCustomView: UIView {

    override func awakeFromNib() {
        setupViews()
    }

    override func prepareForInterfaceBuilder() {
        setupViews()
    }

    func setupViews () {
        backgroundColor = UIColor.redColor()
    }

}
```

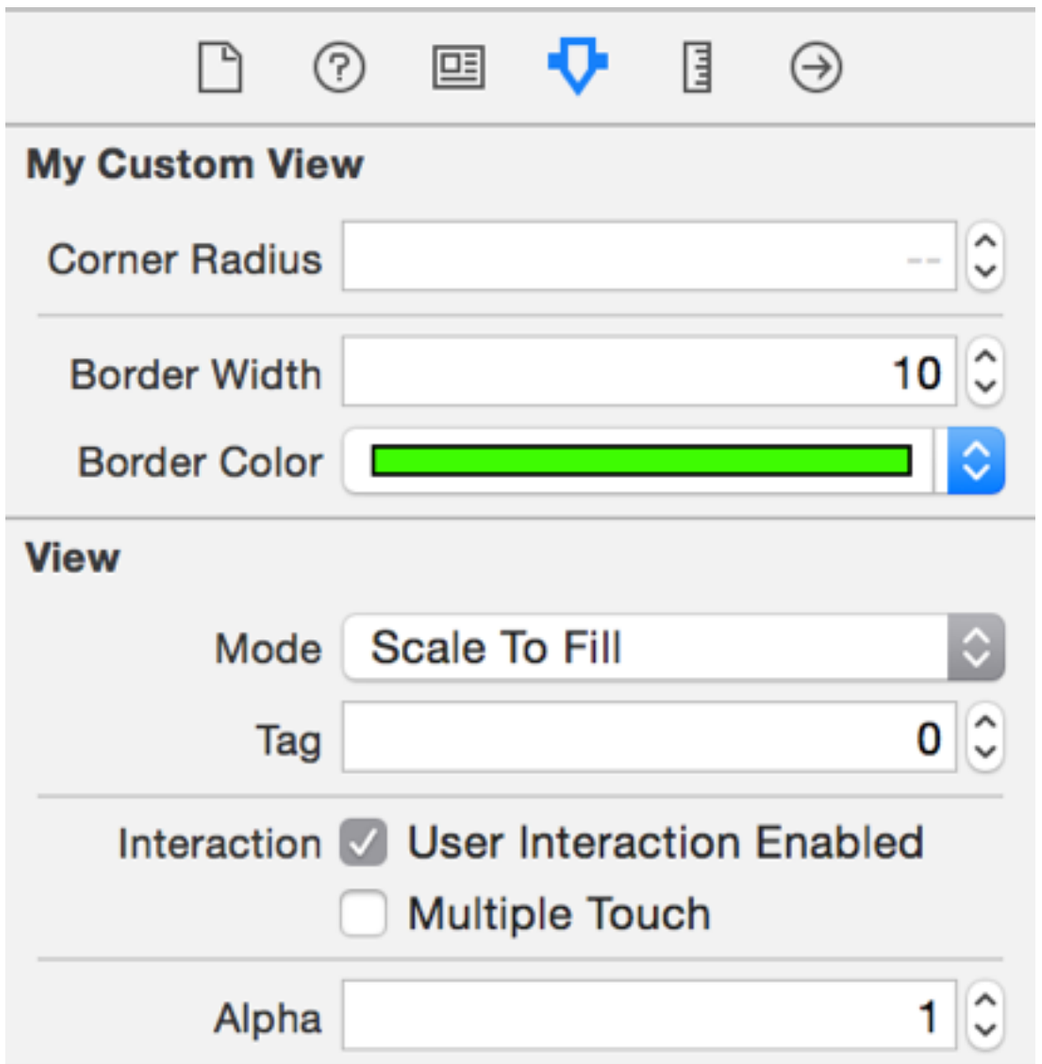
Designable

```
extension UIView {
    public func liveDebugLog(message: String) {
        let logPath = "/tmp/XcodeLiveRendering.log"
        if !NSFileManager.defaultManager().fileExistsAtPath(logPath) {
            NSFileManager.defaultManager().createFileAtPath(logPath, contents:
NSData(), attributes: nil)
        }

        var fileHandle = NSFileHandle(forWritingAtPath: logPath)
        fileHandle!.seekToEndOfFile()

        let date = NSDate()
        let bundle = NSBundle(forClass: self.dynamicType)
        let application: AnyObject =
bundle.objectForInfoDictionaryKey("CFBundleName")!
        let data = "\(date) \(application) \(message)
\n".dataUsingEncoding(NSUTF8StringEncoding, allowLossyConversion: true)
        fileHandle!.writeData(data!)
    }
}
```

Inspectable

- протягивает свойства в IB





Inspectable

- @property (nonatomic) IBInspectable NSInteger lineWidth;

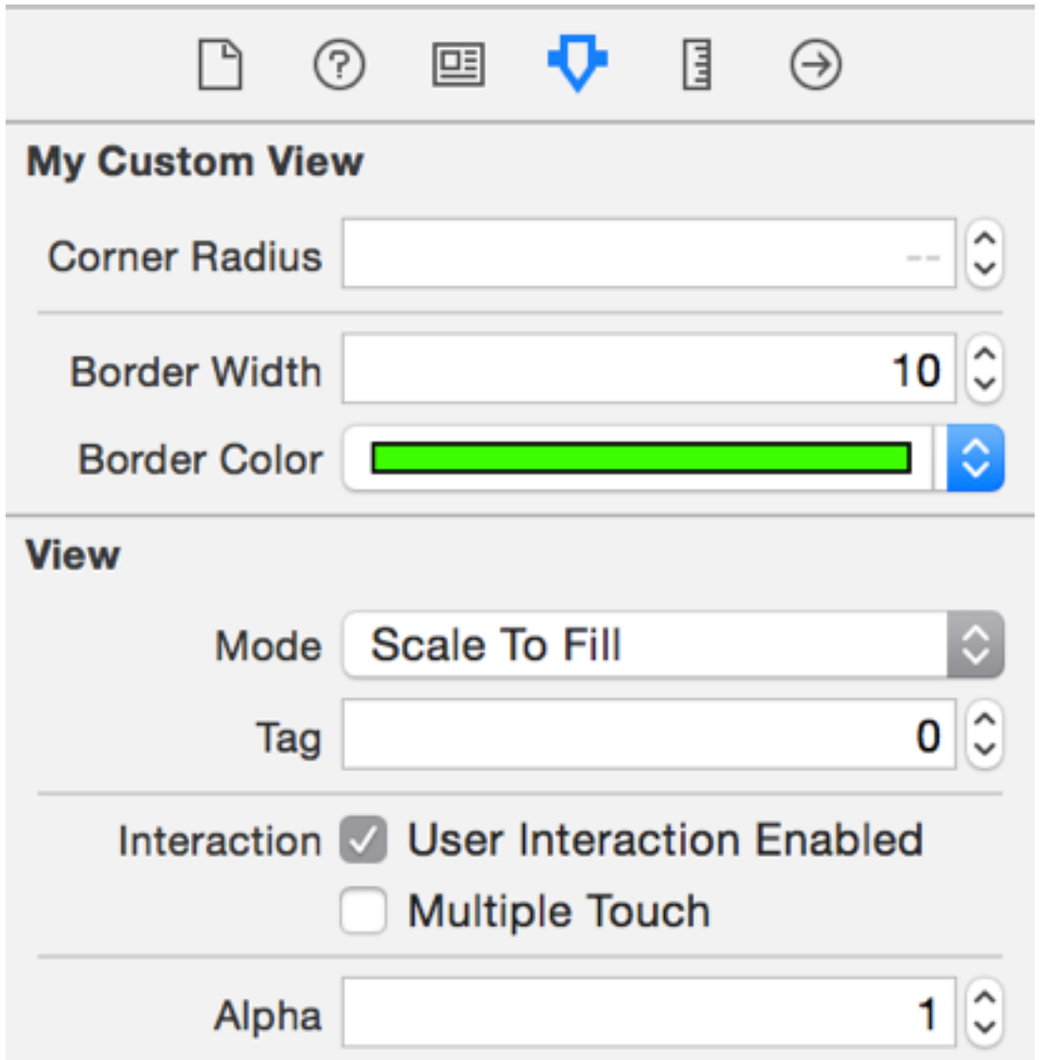
- @IBInspectable var borderWidth: CGFloat = 0 {didSet { layer.borderWidth = borderWidth}}

Inspectable

- Int
- CGFloat
- Double
- String
- Bool
- CGPoint
- CGSize
- CGRect
- UIColor
- UIImage

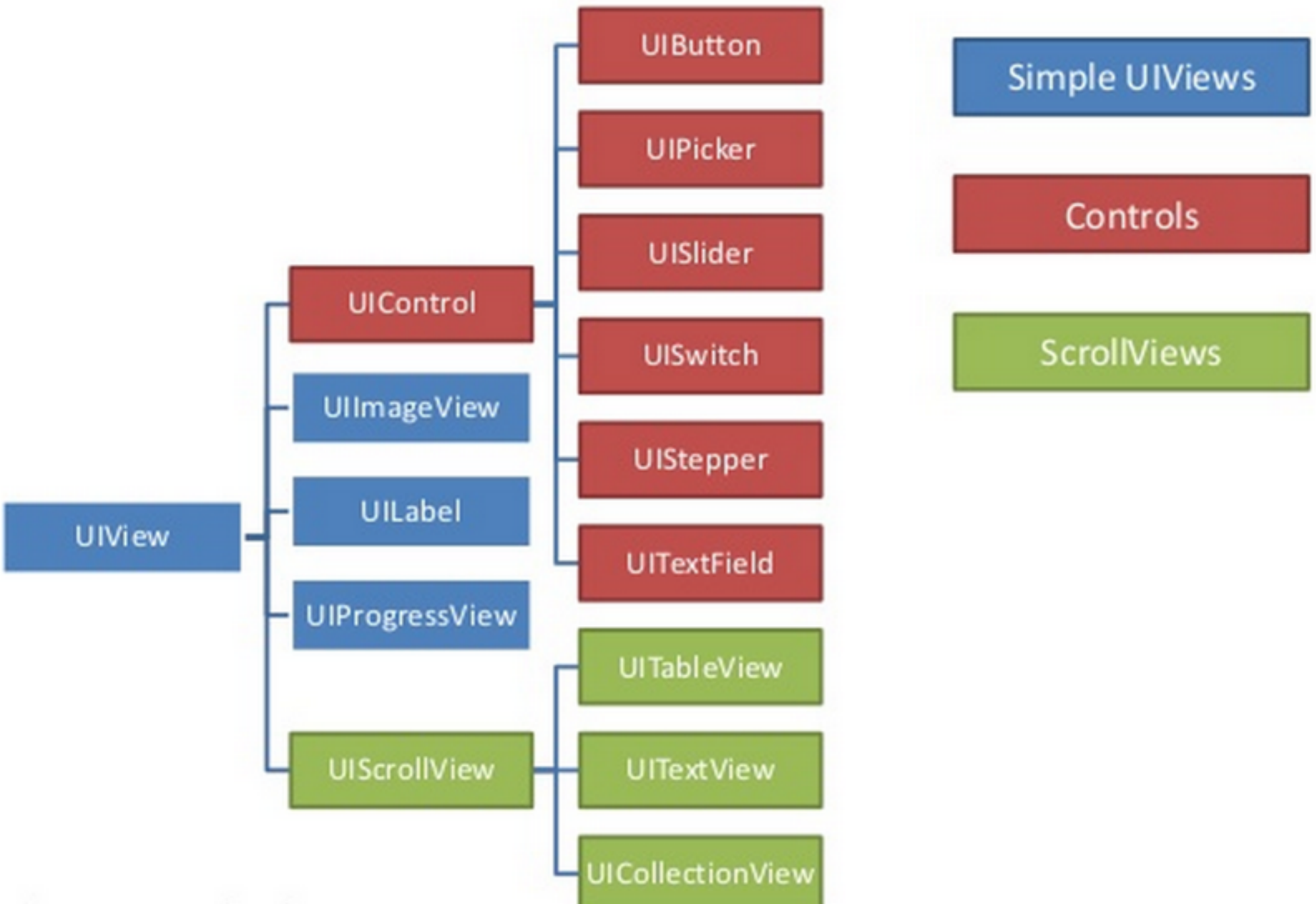
Inspectable

```
@IBDesignable
class MyCustomView: UIView {

    @IBInspectable var cornerRadius: CGFloat = 0 {
        didSet {
            layer.cornerRadius = cornerRadius
            layer.masksToBounds = cornerRadius > 0
        }
    }
    @IBInspectable var borderWidth: CGFloat = 0 {
        didSet {
            layer.borderWidth = borderWidth
        }
    }
    @IBInspectable var borderColor: UIColor? {
        didSet {
            layer.borderColor = borderColor?.CGColor
        }
    }

    override func awakeFromNib() {
        setupViews()
    }

    override func prepareForInterfaceBuilder() {
        setupViews()
    }

    func setupViews () {
        backgroundColor = UIColor.redColor()
    }
}
```



UIView

UIView Subclasses



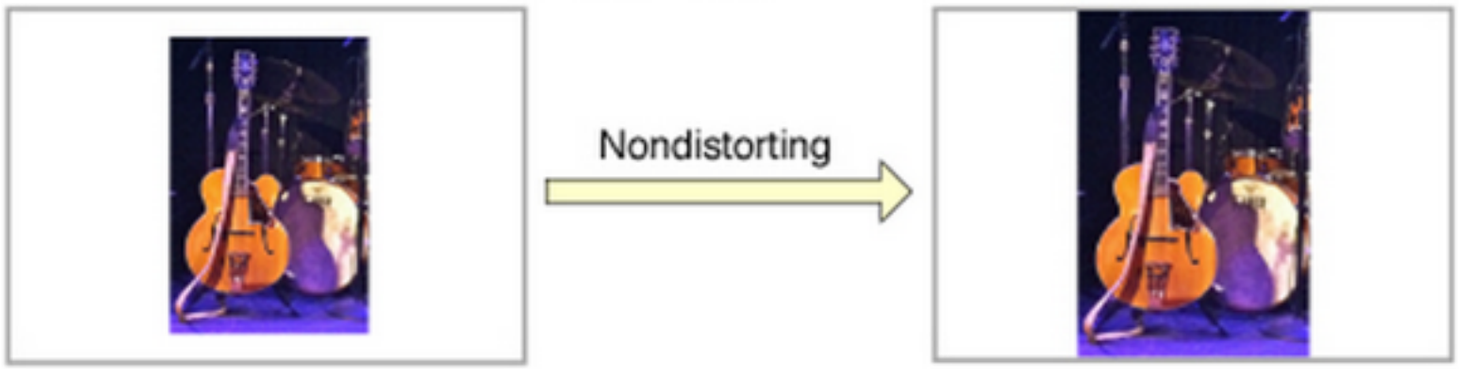
UIView

свойства влияющие на отображение

- frame/bounds
- contentMode
- transform
- …

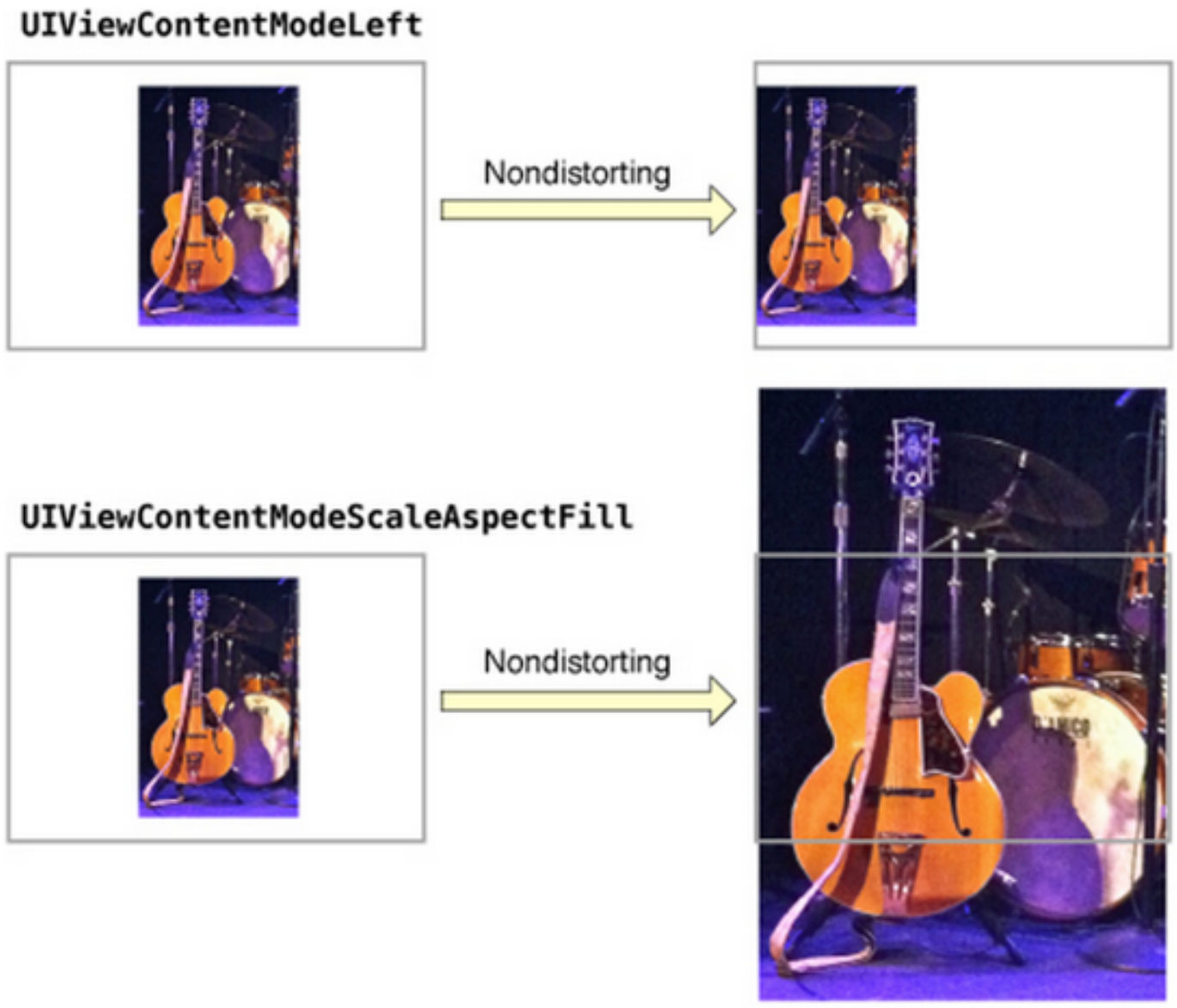
contentMode

`UIViewContentModeScaleAspectFit`

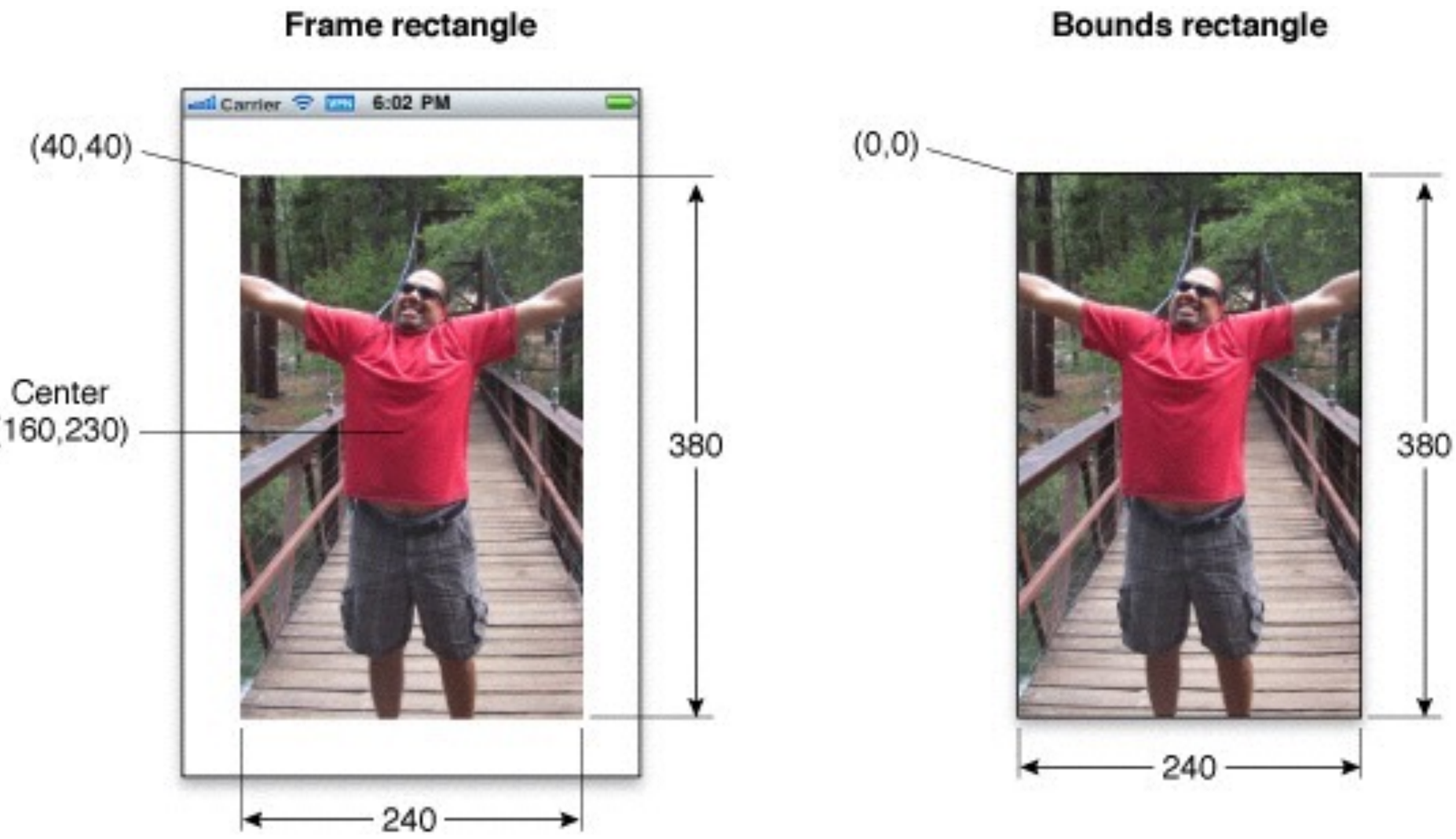


`UIViewContentModeScaleToFill`

contentMode



frame and bounds



transform

```
  scrollView.transform = CGAffineTransformRotate(CGAffineTransformIdentity,
45*(M_PI/180))

 button.transform = CGAffineTransformRotate(CGAffineTransformIdentity, 45*(M_PI/180)
```

Button

UIView

определяют размер view

- sizeThatFits
- sizeToFit
- systemLayoutSizeFittingSize — размер удоволетворяющий констрейнам
- intrinsicContentSize — размер view, основанный на ее свойствах

UILabel

```
func addLabel() {
    let label = UILabel()
    label.frame = CGRectMake(20, 40, 300, 40)
    label.text = "Line1\nLine2"
    label.numberOfLines = 2
    label.sizeToFit()
    view.addSubview(label)
}
```

Line1
Line2

UIImageView

```
func addImageView() {
    let imageView = UIImageView()
    imageView.frame = CGRectMake(20, 40, 300, 140)
    imageView.image = UIImage(named: "sampleImage")
    view.addSubview(imageView)
}
```

UIProgressView

```
    func addProgressView() {
        let progressView = UIProgressView(progressViewStyle:
UIProgressViewStyle.Default)
        progressView.frame = CGRectMake(20, 40, 300, 140)
        progressView.progress = 0.3
        view.addSubview(progressView)
}
```

UIActivityIndicatorView

```
    func addActivityIndicatorView() {
        let indicator = UIActivityIndicatorView(activityIndicatorStyle:
UIActivityIndicatorViewStyle.Gray)
        indicator.frame = CGRectMake(20, 40, 300, 140)
        indicator.startAnimating()
        view.addSubview(indicator)
    }
```

UIPickerView

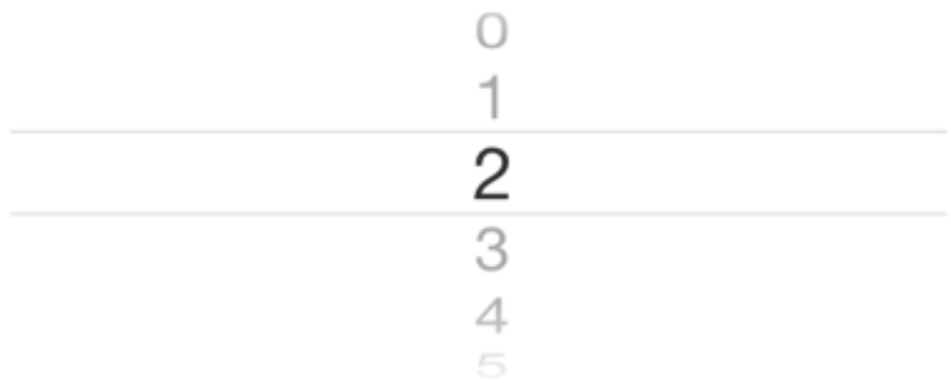
```
protocol UIPickerViewDataSource : NSObjectProtocol {

    // returns the number of 'columns' to display.
    func numberOfComponentsInPickerView(pickerView: UIPickerView) -> Int

    // returns the # of rows in each component..
    func pickerView(pickerView: UIPickerView, numberOfRowsInComponent component:
Int) -> Int
}
```

UIPickerView

```
protocol UIPickerViewDelegate : NSObjectProtocol {

    // returns width of column and height of row for each component.
    optional func pickerView(pickerView: UIPickerView, widthForComponent component: Int) -> CGFloat
    optional func pickerView(pickerView: UIPickerView, rowHeightForComponent component: Int) -> CGFloat

    // these methods return either a plain NSString, a NSAttributedString, or a view (e.g UILabel) to
display the row for the component.
    // for the view versions, we cache any hidden and thus unused views and pass them back for reuse.
    // If you return back a different object, the old one will be released. the view will be centered in
the row rect
    optional func pickerView(pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int)
-> String!
    @availability(iOS, introduced=6.0)
    optional func pickerView(pickerView: UIPickerView, attributedTitleForRow row: Int, forComponent
component: Int) -> NSAttributedString? // attributed title is favored if both methods are implemented
    optional func pickerView(pickerView: UIPickerView, viewForRow row: Int, forComponent component: Int,
reusingView view: UIView!) -> UIView

    optional func pickerView(pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int)
}
```

UIPickerView

```
func addPickerView() {
    let pickerView = UIPickerView()
    pickerView.frame = CGRectMake(20, 40, 300, 140)
    pickerView.dataSource = self
    pickerView.delegate = self
    view.addSubview(pickerView)
}

func pickerView(pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) -> String!
    return "\(row)"
}

func numberOfComponentsInPickerView(pickerView: UIPickerView) -> Int {
    return 1
}

func pickerView(pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int
{
    return 10
}
```

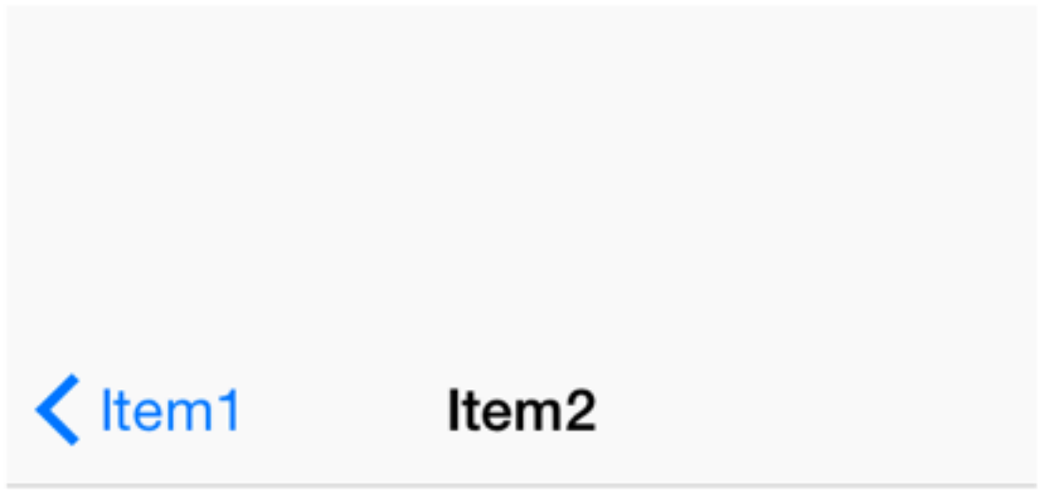


UINavigationBar

```
protocol UINavigationBarDelegate : UIBarPositioningDelegate, NSObjectProtocol
{

    optional func navigationBar(navigationBar: UINavigationBar,
shouldPushItem item: UINavigationItem) -> Bool // called to push. return NO
not to.
    optional func navigationBar(navigationBar: UINavigationBar, didPushItem
item: UINavigationItem) // called at end of animation of push or immediately
if not animated
    optional func navigationBar(navigationBar: UINavigationBar, shouldPopItem
item: UINavigationItem) -> Bool // same as push methods
    optional func navigationBar(navigationBar: UINavigationBar, didPopItem
item: UINavigationItem)
}
```

UINavigationBar

```
func addNavigationBar() {
    let navBar = UINavigationBar()
    navBar.frame = CGRectMake(20, 40, 300, 140)
    navBar.pushNavigationItem(UINavigationItem(title: "Item1"), animated: false)
    navBar.pushNavigationItem(UINavigationItem(title: "Item2"), animated: false)
    navBar.delegate = self
    view.addSubview(navBar)
    }

    func navigationBar(navigationBar: UINavigationBar, shouldPopItem item:
UINavigationItem) -> Bool {
        println("NavigationBar should pop item \(item)")
        return true
    }
```



UITabBar

```
protocol UITabBarDelegate : NSObjectProtocol {

    optional func tabBar(tabBar: UITabBar, didSelectItem item: UITabBarItem!) //
called when a new view is selected by the user (but not programatically)

    /* called when user shows or dismisses customize sheet. you can use the
'willEnd' to set up what appears underneath.
     changed is YES if there was some change to which items are visible or which
order they appear. If selectedItem is no longer visible,
     it will be set to nil.
     */

    optional func tabBar(tabBar: UITabBar, willBeginCustomizingItems items:
[AnyObject]) // called before customize sheet is shown. items is current item list
    optional func tabBar(tabBar: UITabBar, didBeginCustomizingItems items:
[AnyObject]) // called after customize sheet is shown. items is current item list
    optional func tabBar(tabBar: UITabBar, willEndCustomizingItems items:
[AnyObject], changed: Bool) // called before customize sheet is hidden. items is new
item list
    optional func tabBar(tabBar: UITabBar, didEndCustomizingItems items:
[AnyObject], changed: Bool) // called after customize sheet is hidden. items is new
item list
}
```

UITabBar

```
func addTabBar() {
    let tabBar = UITabBar()
    tabBar.frame = CGRectMake(20, 40, 300, 140)
    tabBar.items = [
        UITabBarItem(title: "item1", image: nil, tag: 0),
        UITabBarItem(title: "item2", image: nil, tag: 0)
    ]
    tabBar.delegate = self
    view.addSubview(tabBar)
}

func tabBar(tabBar: UITabBar, didSelectItem item: UITabBarItem!) {
    println("TabBar select item \(tabBar)")
}
```

UIControl

UIControl

```
class UIControl : UIView {

    var enabled: Bool // default is YES. if NO, ignores touch events and subclasses may draw
differently
    var selected: Bool // default is NO may be used by some subclasses or by application
    var highlighted: Bool // default is NO. this gets set/cleared automatically when touch
enters/exits during tracking and cleared on up
    var contentVerticalAlignment: UIControlContentVerticalAlignment // how to position
content vertically inside control. default is center
    var contentHorizontalAlignment: UIControlContentHorizontalAlignment // how to position
content hozontally inside control. default is center

    // add target/action for particular event. you can call this multiple times and you can
specify multiple target/actions for a particular event.
    // passing in nil as the target goes up the responder chain. The action may optionally
include the sender and the event in that order
    // the action cannot be NULL. Note that the target is not retained.
    func addTarget(target: AnyObject?, action: Selector, forControlEvents controlEvents:
UIControlEvents)

    // remove the target/action for a set of events. pass in NULL for the action to remove
all actions for that target
    func removeTarget(target: AnyObject?, action: Selector, forControlEvents controlEvents:
UIControlEvents)
```

UIControlEvents

```
struct UIControlEvents : RawOptionSetType {
    init(_ rawValue: UInt)
    init(rawValue: UInt)

    static var TouchDown: UIControlEvents { get } // on all touch downs
    static var TouchDownRepeat: UIControlEvents { get } // on multiple touchdowns (tap count > 1)
    static var TouchDragInside: UIControlEvents { get }
    static var TouchDragOutside: UIControlEvents { get }
    static var TouchDragEnter: UIControlEvents { get }
    static var TouchDragExit: UIControlEvents { get }
    static var TouchUpInside: UIControlEvents { get }
    static var TouchUpOutside: UIControlEvents { get }
    static var TouchCancel: UIControlEvents { get }

    static var ValueChanged: UIControlEvents { get } // sliders, etc.

    static var EditingDidBegin: UIControlEvents { get } // UITextField
    static var EditingChanged: UIControlEvents { get }
    static var EditingDidEnd: UIControlEvents { get }
    static var EditingDidEndOnExit: UIControlEvents { get } // 'return key' ending editing

    static var AllTouchEvents: UIControlEvents { get } // for touch events
    static var AllEditingEvents: UIControlEvents { get } // for UITextField
    static var ApplicationReserved: UIControlEvents { get } // range available for application use
    static var SystemReserved: UIControlEvents { get } // range reserved for internal framework use
    static var AllEvents: UIControlEvents { get }
}
```

UIButton

```
func addButton() {
        let button: UIButton = UIButton.buttonWithType(UIButtonType.Custom) as!
UIButton
        button.frame = CGRectMake(20, 40, 100, 40)
        button.setTitleColor(UIColor.redColor(), forState: UIControlState.Normal)
        button.setTitle("Button", forState: UIControlState.Normal)
        button.setTitle("Highlighted", forState: .Highlighted)
        button.addTarget(self, action: "buttonTapped:", forControlEvents:
UIControlEvents.TouchUpInside)
        view.addSubview(button)
    }

    func buttonTapped(button: UIButton!) {
        println("Button tapped \(button)")
    }
```

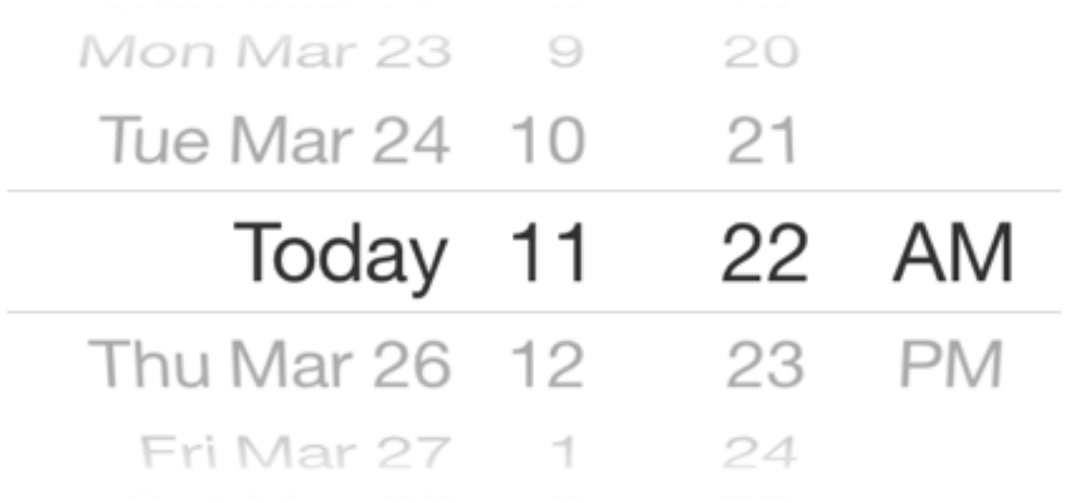
UISwitch

```
    func addSwitch() {
        let sw = UISwitch()
        sw.frame = CGRectMake(20, 40, 100, 40)
        sw.addTarget(self, action: "switchTapped:", forControlEvents:
UIControlEvents.TouchUpInside)
        sw.addTarget(self, action: "switchChanged:", forControlEvents:
UIControlEvents.ValueChanged)
        view.addSubview(sw)
    }

    func switchChanged(sw: UISwitch!) {
        println("Switch changed \(sw.on)")
    }

    func switchTapped(sw: UISwitch!) {
        println("Switch tapped \(sw.on)")
    }
```

UIDatePicker

```
    func addDatePicker() {
        let picker = UIDatePicker()
        picker.frame = CGRectMake(20, 40, 300, 40)
        picker.addTarget(self, action: "pickerChanged:", forControlEvents:
UIControlEvents.ValueChanged)
        view.addSubview(picker)
    }
```

```
func pickerChanged(picker: UIDatePicker!) {
    println("Picker changed \(picker.date)")
}
```



UIStepper

```
    func addStepper() {
        let stepper = UIStepper()
        stepper.frame = CGRectMake(20, 40, 300, 40)
        stepper.addTarget(self, action: "stepperChanged:", forControlEvents:
UIControlEvents.ValueChanged)
        view.addSubview(stepper)
    }
```

```
func stepperChanged(stepper: UIStepper!) {
    println("Stepper changed \(stepper.value)")
}
```

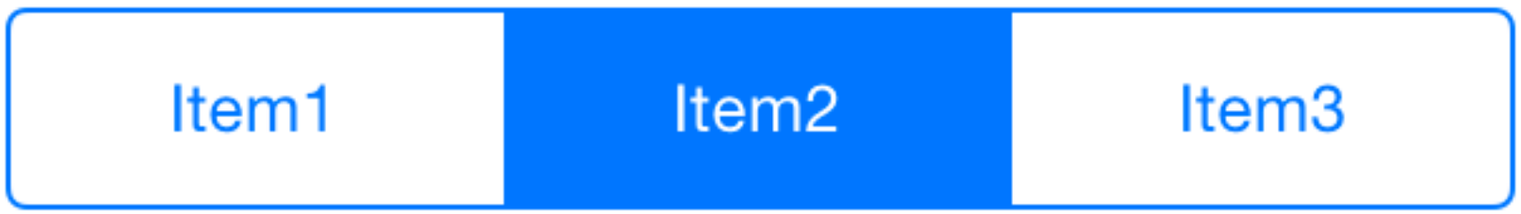
UISlider

```
    func addSlider() {
        let slider = UISlider()
        slider.frame = CGRectMake(20, 40, 300, 40)
        slider.addTarget(self, action: "sliderChanged:", forControlEvents:
UIControlEvents.ValueChanged)
        view.addSubview(slider)
    }

    func sliderChanged(slider: UISlider!) {
        println("Slider changed \(slider.value)")
    }
```

UISegmentControl

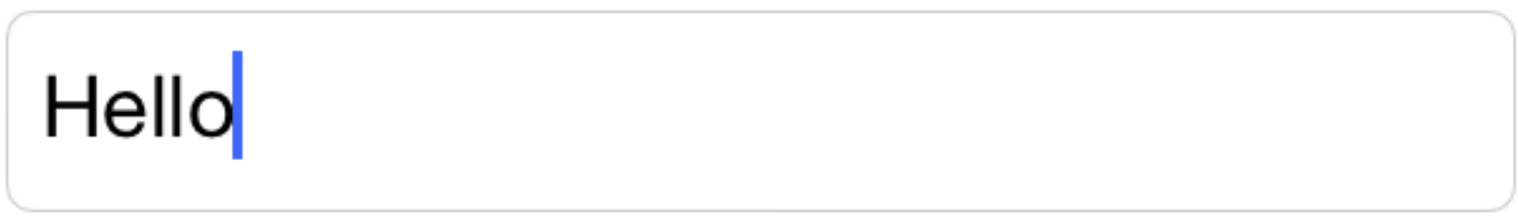
```
    func addSegmentControl() {
        let segment = UISegmentedControl(items: ["Item1", "Item2", "Item3"])
        segment.frame = CGRectMake(20, 40, 300, 40)
        segment.addTarget(self, action: "segmentChanged:", forControlEvents:
UIControlEvents.ValueChanged)
        view.addSubview(segment)
    }
```

```
func segmentChanged(segment: UISegmentedControl!) {
    println("Segment changed \(segment.selectedSegmentIndex)")
}
```



UITextField

```
    func addTextField() {
        let textField = UITextField()
        textField.frame = CGRectMake(20, 40, 300, 40)
        textField.borderStyle = UITextBorderStyle.RoundedRect
        textField.addTarget(self, action: "textFieldChanged:", forControlEvents:
UIControlEvents.EditingChanged)
        view.addSubview(textField)
    }

func textFieldChanged(textField: UITextField!) {
    println("Text changed \(textField.text)")
}
```



UIPageControl

```
    func addPageControl() {
        let pageControl = UIPageControl()
        pageControl.frame = CGRectMake(20, 40, 300, 140)
        pageControl.numberOfPages = 3
        pageControl.addTarget(self, action: "pageControlChanged:", forControlEvents:
UIControlEvents.ValueChanged)
        view.addSubview(pageControl)
        view.backgroundColor = UIColor.blackColor()
}

func pageControlChanged(pageControl: UIPageControl!) {
    println("Page changed \(pageControl.currentPage)")
}
```

UIScrollView

UIScrollView

- по сути обычная UIView с UIPanGestureRecognizer, на который навешено обновление frame и UIPinchGestureRecognizer на который навешен scale

UIScrollView

```
protocol UIScrollViewDelegate : NSObjectProtocol {

    optional func scrollViewDidScroll(scrollView: UIScrollView) // any offset changes
    @availability(iOS, introduced=3.2)
    optional func scrollViewDidZoom(scrollView: UIScrollView) // any zoom scale changes

    // called on start of dragging (may require some time and or distance to move)
    optional func scrollViewWillBeginDragging(scrollView: UIScrollView)
    // called on finger up if the user dragged. velocity is in points/millisecond. targetContentOffset may be changed to
adjust where the scroll view comes to rest
    @availability(iOS, introduced=5.0)
    optional func scrollViewWillEndDragging(scrollView: UIScrollView, withVelocity velocity: CGPoint, targetContentOffset:
UnsafeMutablePointer<CGPoint>)
    // called on finger up if the user dragged. decelerate is true if it will continue moving afterwards
    optional func scrollViewDidEndDragging(scrollView: UIScrollView, willDecelerate decelerate: Bool)

    optional func scrollViewWillBeginDecelerating(scrollView: UIScrollView) // called on finger up as we are moving
    optional func scrollViewDidEndDecelerating(scrollView: UIScrollView) // called when scroll view grinds to a halt

    optional func scrollViewDidEndScrollingAnimation(scrollView: UIScrollView) // called when setContentOffset/
scrollRectVisible:animated: finishes. not called if not animating

    optional func viewForZoomingInScrollView(scrollView: UIScrollView) -> UIView? // return a view that will be scaled. if
delegate returns nil, nothing happens
    @availability(iOS, introduced=3.2)
    optional func scrollViewWillBeginZooming(scrollView: UIScrollView, withView view: UIView!) // called before the scroll
view begins zooming its content
    optional func scrollViewDidEndZooming(scrollView: UIScrollView, withView view: UIView!, atScale scale: CGFloat) //
scale between minimum and maximum. called after any 'bounce' animations

    optional func scrollViewShouldScrollToTop(scrollView: UIScrollView) -> Bool // return a yes if you want to scroll to
the top. if not defined, assumes YES
    optional func scrollViewDidScrollToTop(scrollView: UIScrollView) // called when scrolling animation finished. may be
called immediately if already at top
}
```

UIScrollView

- scrollViewShouldScrollToTop срабатывает при тапе на статус бар

- если есть больше одной UIScrollView с scrollsToTop, то не будет работать

UIScrollView

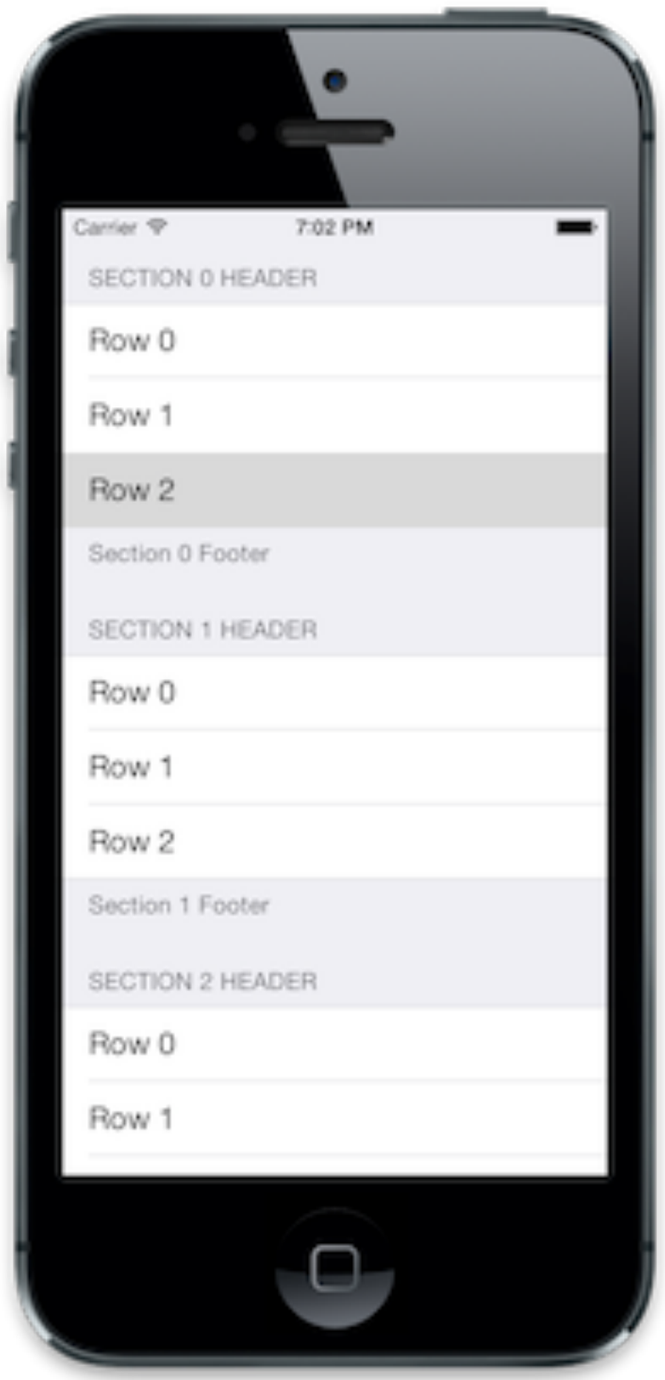
```
func addScrollView() {
    let scrollView = UIScrollView()
    scrollView.frame = CGRectMake(20, 240, 300, 140)
    scrollView.backgroundColor = UIColor.lightGrayColor()
    scrollView.delegate = self

    let imageView = UIImageView()
    imageView.image = UIImage(named: "sampleImage")
    imageView .sizeToFit()

    scrollView.contentSize = imageView.frame.size
    scrollView.addSubview(imageView)
    scrollView.minimumZoomScale = 0.1
    scrollView.maximumZoomScale = 3.0
    scrollView.zoomScale = 1.0

    view.addSubview(scrollView)
}

func viewForZoomingInScrollView(scrollView: UIScrollView) -> UIView? {
    return scrollView.subviews[0] as? UIView
}
```

UITableView

- можно сказать, что это умный UIScrollView
- умеет спрашивать контент (ячейки) для определенного смещения по вертикали
- контент (ячейки) представлен при помощи UITableViewCell
- элементы, которые находятся за видимыми границами удаляются из иерархии (иначе будут потери производительности для больших таблиц)

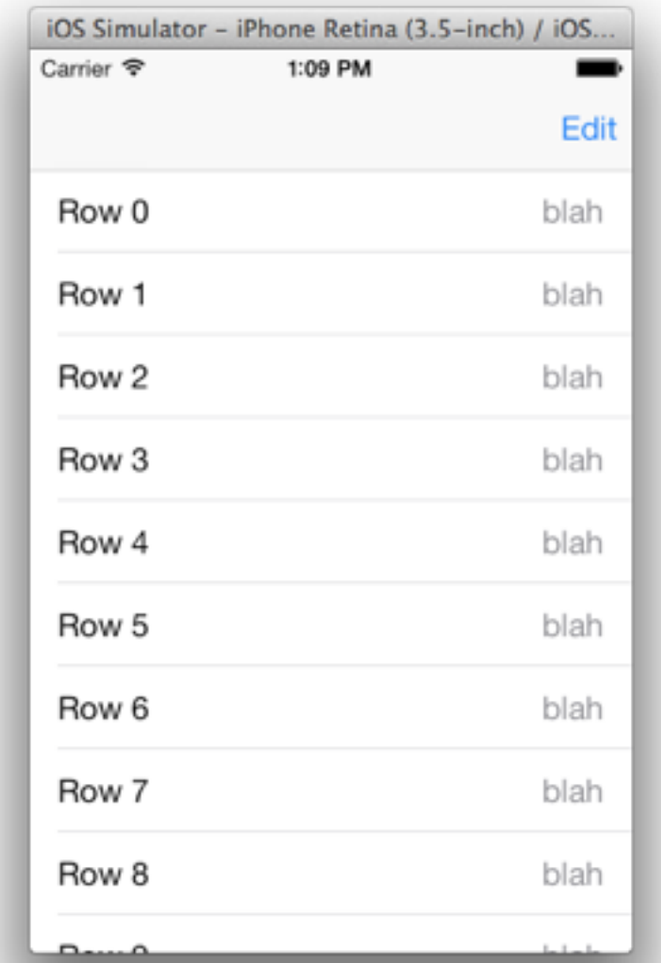
UITableView

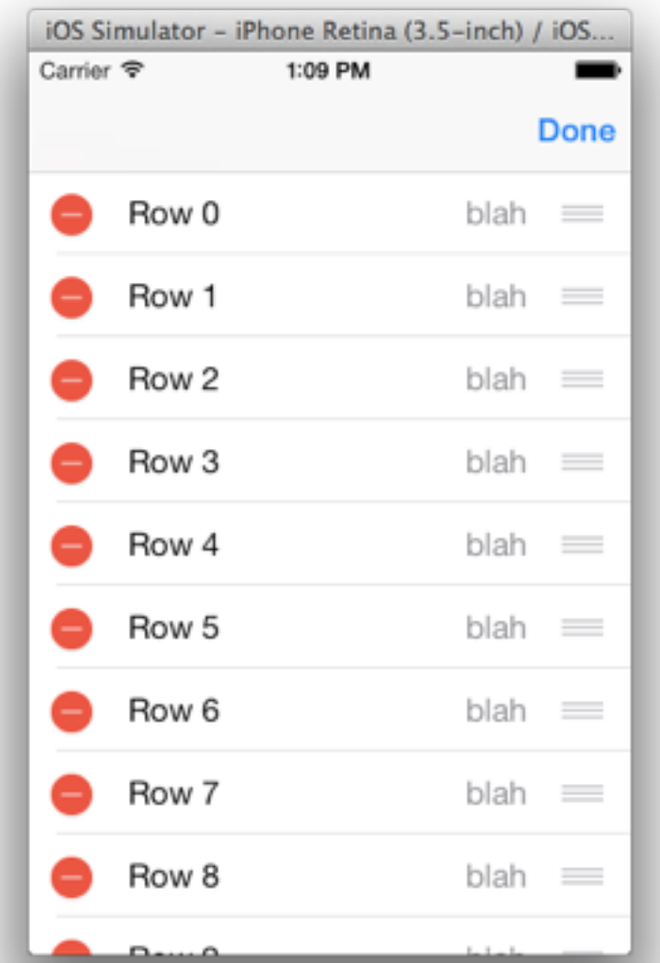
- можно делать заголовки для секций (UIView)



UITableView

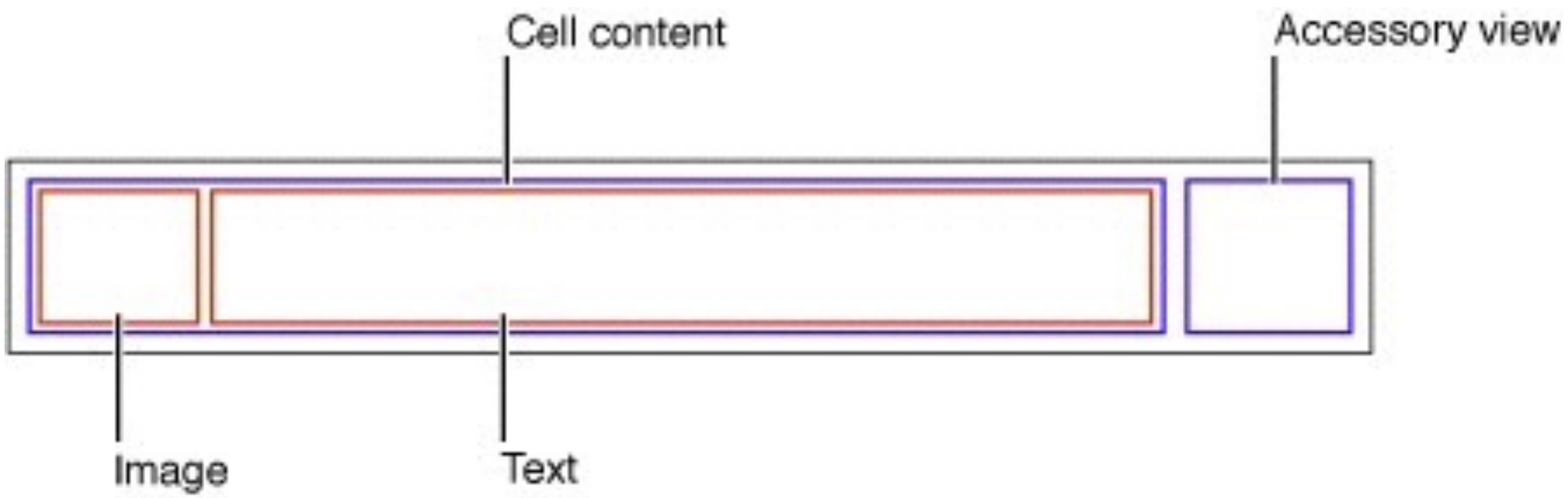
- можно редактировать таблицу (удалять и передвигать ячейки)





UITableViewCell

- обычная UIView с набором subviews и рядом полезных для UITableView свойств и методов (reuseIdentifier, prepareForReuse, …)



UICollectionView

- можно сказать, что это умный UIScrollView

- в отличие от UITableView может располагать ячейки как угодно

UICollectionViewLayout

- управляет положением элементов

- элементы можно накладывать друг на друга

Animations

CALayer

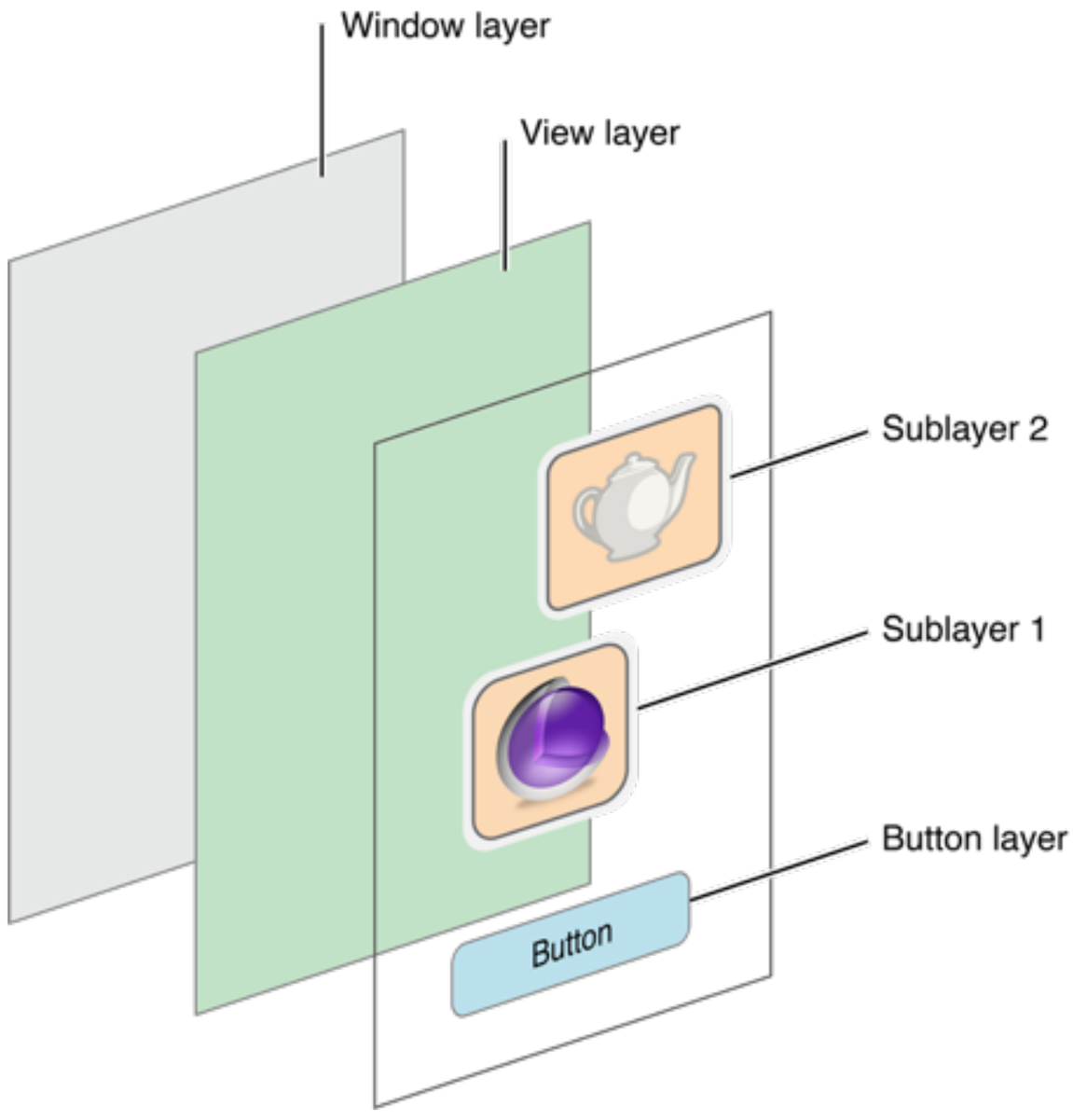
- занимается отображением контента и анимациями

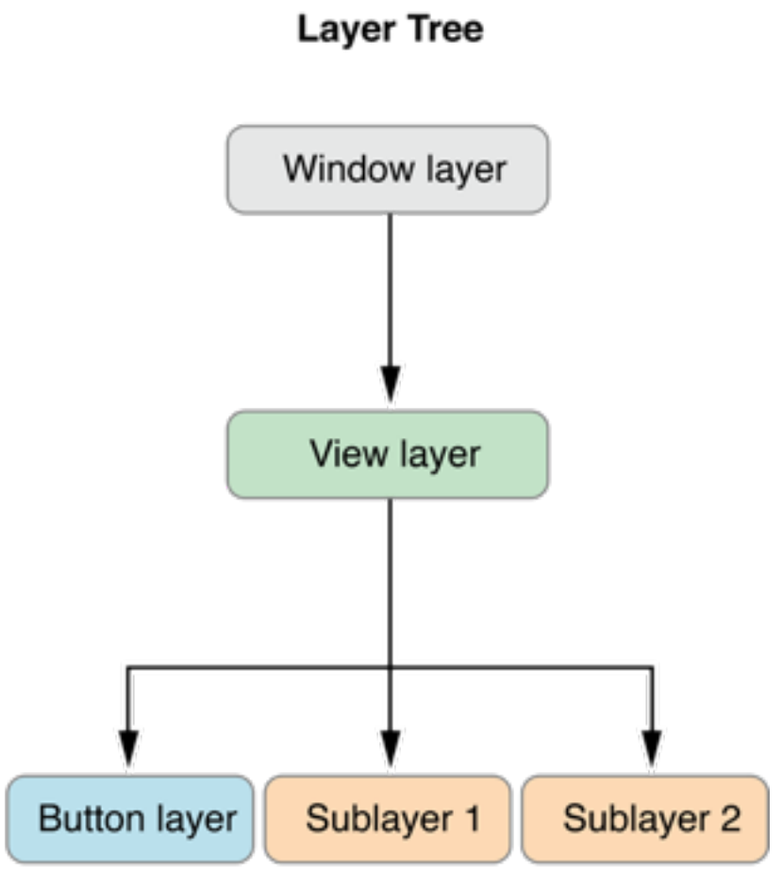
- у каждой UIView есть CALayer

- UIView только конфигурируете CALayer и предоставляет контент для отображения (drawRect)

- констрейнами (NSLayoutConstraint) управляет UIView

CALayer


Window layer
View layer
Sublayer 2
Sublayer 1
Button layer
Button


Layer Tree
Window layer
View layer
Button layer
Sublayer 1
Sublayer 2

CALayer

- CAGradientLayer
- CAShapeLayer
- CAScrollLayer
- CATiledLayer
- …

CAGradientLayer

```
let l = CAGradientLayer()
l.frame = CGRectMake(100, 100, 100, 100)
l.colors = [UIColor.redColor().CGColor, UIColor.greenColor().CGColor]
l.locations = [ 0.0, 1.0]
self.view.layer.addSublayer(l)
```

CAlayer

- по умолчанию изменения все свойств анимировано, но UIView отключает анимацию

UIView

у UIView анимируются

- frame
- bounds
- center
- transform
- alpha
- backgroundColor
- contentStretch

UIView

```
func buttonTapped(button: UIButton!) {
        println("Button tapped \(button)")

        UIView.animateWithDuration(
            1,
            delay: 0,
            options: UIViewAnimationOptions.CurveLinear,
            animations: { () -> Void in
                button.transform =
CGAffineTransformRotate(CGAffineTransformIdentity, CGFloat(M_PI))
            })
            { (value: Bool) -> Void in
                UIView.animateWithDuration(1, animations: { () -> Void in
                    button.transform = CGAffineTransformIdentity
                })
        }
    }
```

CABasicAnimation

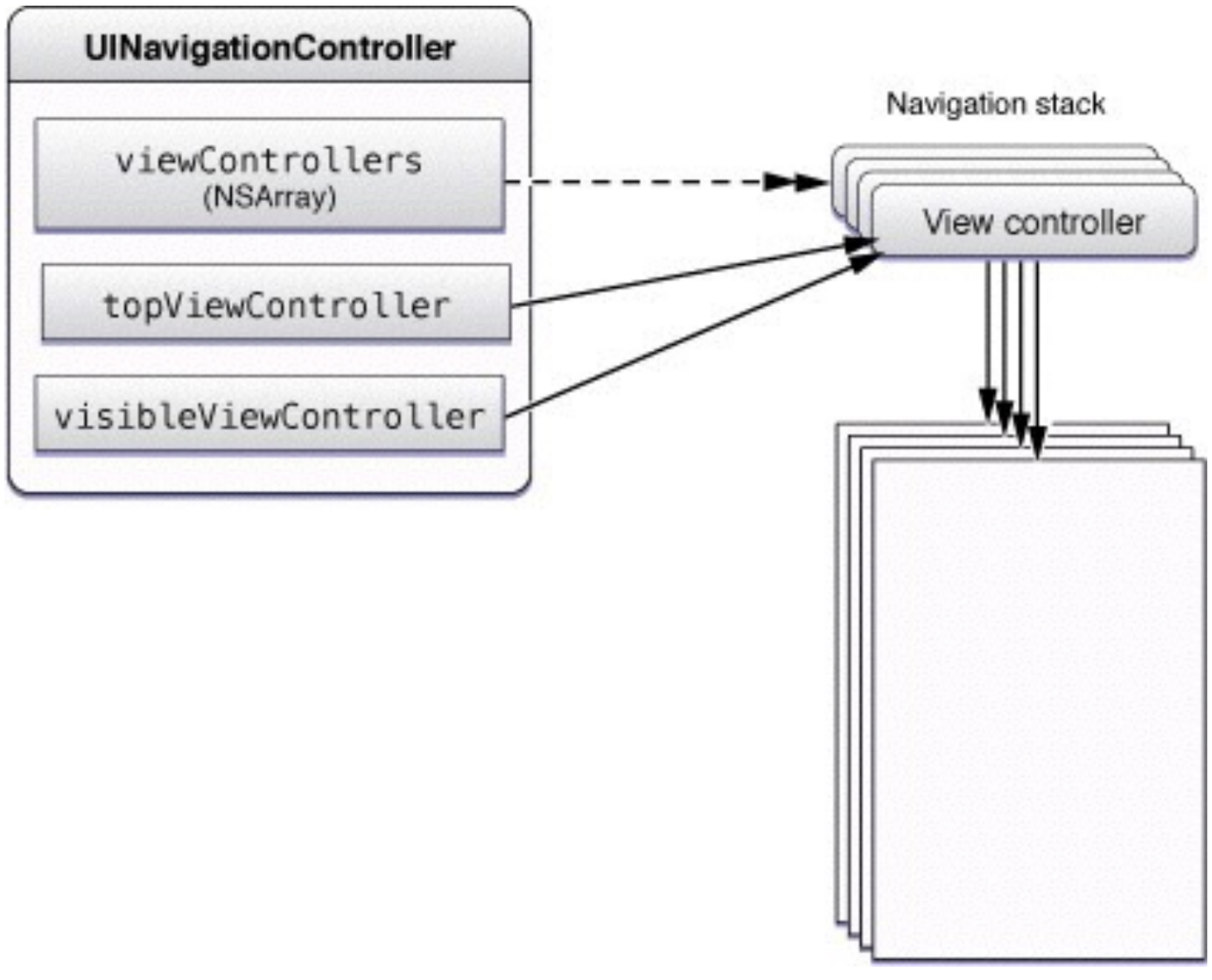
```
let rotateAnimation = CABasicAnimation(keyPath: "transform.rotation")
 rotateAnimation.fromValue = 0.0
 rotateAnimation.toValue = CGFloat(M_PI)
 rotateAnimation.duration = 1
 button.layer.addAnimation(rotateAnimation, forKey: nil)
```

Controller

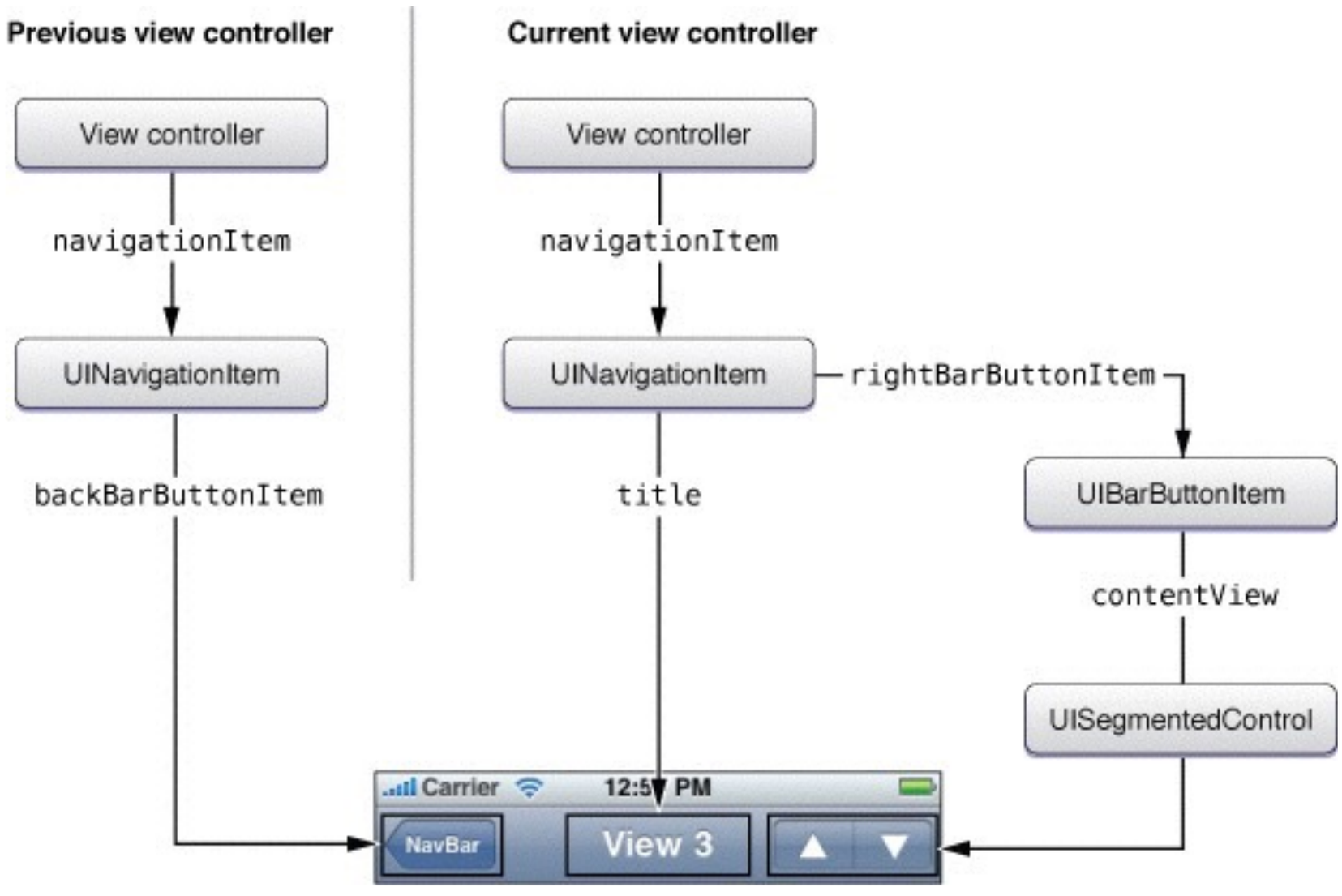
Modal

```
let vc = UIViewController()
vc.modalPresentationStyle = UIModalPresentationStyle.FormSheet
self.presentViewController(vc, animated: true, completion: nil)
```

UINavigationController



UINavigationBar



UINavigationController

```
var vc = ViewController()
let navController = UINavigationController(rootViewController: vc)
self.presentViewController(navController, animated: true) { () -> Void in
    navController.pushViewController(ViewController(), animated: true)
}
```