Object Oriented Programming

A tenant of good source code is something that is reusable and easy to read. If you can create something once and reuse it, you can:

1. Write less code
2. Have an easier time maintaining your code base
3. Reuse your code in other projects

Writing code using the principles of OOP will allow you to accomplish the above tasks while maintaining logical separation of roles between your objects.

> Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which are data structures that contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods. [1]

Terms

Let's talk about the ways Swift can help you rock out OOP.

Defining a class

And organize your code into reusable objects with a class:

```
class ExampleClass {

    var memberVariable:String

    init() {
        self.memberVariable = "Initial value."
    }

    func exampleFunction(thisIsFun:Bool) -> String {

        var returnValue = "No, this isn't fun."
        if thisIsFun {
            returnValue = "Yes, this is fun!"
        }

        return returnValue
    }
}

var exampleClass = ExampleClass()
print("We created a class and its member variable contains \(exampleClass.memberVariable) )"
```

Inheritance

A class can inherit another class and therefore capture its methods as its own. A subclass inherits the superclass.

```
import UIKit

class YourViewController: UIViewController {

}
```

In this simple example above, we are using Swift to describe YourViewController which inherits all the methods of UIViewController. When we inherit the superclass, we can not only obtain functionality but can override that functionality as we want.

```
import UIKit

class YourViewController: UIViewController {
    override func viewWillAppear(animated: Bool) {
        view.backgroundColor = UIColor.blackColor()
    }
}
```

Protocols

Protocols are similar to inheritance in that we obtain a set of functions and properties through declaration. They differ in that you are required to define these methods when implementing the protocol.

```
import Foundation

// Defines a protocol
protocol ExampleProtocol {

    var maxValue:Int {get set}

    func generateRandomNumber() -> Int
}

// Defines the class which implements the protocol
class ExampleImplementation: ExampleProtocol {

    var maxValue:Int

    init (initialMaximumValue:Int) {
        maxValue = initialMaximumValue
    }

    func generateRandomNumber() -> Int {
        return random() % maxValue;
    }
}

// Example usage
let example = ExampleImplementation(initialMaximumValue: 10)
example.generateRandomNumber()
```

A real-world example, extending our previous, would be to conform YourViewController to the UITableViewDataSource protocol:

```
import UIKit

class YourViewController: UIViewController, UITableViewDataSource {

    ///////////
    // Inheritance
    override func viewWillAppear(animated: Bool) {
        view.backgroundColor = UIColor.blackColor()
    }

    ///////////
    // Protocol
    // Required: filling out protocol functions

    // protocol definition 1
    func numberOfSectionsInTableView(tableView: UITableView) -> Int {
        return 1
    }

    // protocol definition 2
    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return 1
    }

    // protocol definition 3
    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {

        var tableCell = UITableViewCell()
```

```
        return tableCell
    }
}
```

Polymorphism

Polymorphism allows you to write more generic code that works with families of objects, rather than writing code for a specific class. [2]

```
var x = UILabel()
var y = UITextView()
var z = UIButton()

func updateTheTag(control:UIView, updatedTag:Int) {
    control.tag = updatedTag
}

updateTheTag(x, 1)
updateTheTag(y, 2)
updateTheTag(z, 3)
```

In this above example, UILabel, UITextView and UIButton all derive from the base class of UIView and that base class defines a tag property. Although each of these classes may have a specific purpose, they all can access their base classes properties in a generic way.

Swift types

Enumerations

An enumeration defines a common type for a group of related values and enables you to work with those values in a type-safe way within your code. [3]

```
enum CompassPoint {
    case North
    case South
    case East
    case West
}

// Int type
enum AgesOfChildren:Int {
    case John = 20
    case Jacob = 25
    case Jingleheimer = 30
    case Schmidt = 35
}

// String type
enum SectionHeader : String {
    case FirstColumn = "Column 1"
    case SecondColumn = "Column 2"
    case ThirdColumn = "Column 3"
    case FourthColumn = "Column 4"
    case FifthColumn = "Column 5"
}
```

Structures vs Classes

Apples documentation goes into detail defining structures and classes. That resource is listed as number 5 below, and is summarized here.

> Classes and structures are general-purpose, flexible constructs that become the building blocks of your program's code.

> Similarities:
>
> - Define properties to store values
> - Define methods to provide functionality
> - Define subscripts to provide access to their values using subscript syntax
> - Define initializers to set up their initial state
> - Be extended to expand their functionality beyond a default implementation
> - Conform to protocols to provide standard functionality of a certain kind

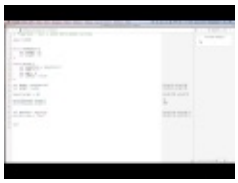
Structures

```
struct Beer {
    var name:String
    var ounces:Float
    var alcoholContent:Float
    var bitterness:Int
}

var stoneIPA = Beer(name:"Stone IPA", ounces:22.0, alcoholContent:6.9, bitterness:85)
```

[Video] Value versus References

When talking about classes there is one key difference and that is **value** versus **reference**. Structures are passed by value where classes are passed by reference.



Classes

Now classes are similar to structures with a few exceptions, including being passed as reference types. Classes also allow for inheritance, which enables us to define more complex object schemes, such as the following:

```
class Human {
    var name:String
    var type:String
    var age:Int
    var spouse:Human?

    init(name:String, type:String, age:Int) {
        self.name = name
        self.type = type
        self.age = age
    }

    func becomeMarried(spouse:Human) -> () {
        self.spouse = spouse
    }
}

let me = Human(name: "Jeff", type: "Human", age: 34)
me.spouse

let wife = Human(name: "Katie", type: "Human", age: 32)
me.becomeMarried(wife)
wife.spouse

class Baby : Human {
    var diaperChangesToday:Int

    override init(name: String, type: String, age: Int) {
```

```
        diaperChangesToday = 0
        super.init(name: name, type: type, age: age)
    }

    func incrementDiaperChanges() -> String {
        diaperChangesToday++
        return("\(diaperChangesToday) diapers? C'mon now!")
    }
}

let newBaby = Baby(name: "Raleigh", type: "Human", age: 0)
newBaby.incrementDiaperChanges()
newBaby.incrementDiaperChanges()
newBaby.incrementDiaperChanges()
```

References

An Introduction to Object-Oriented Programming - http://blog.codeclimate.com/blog/2014/06/19/oo-swift/

[1] Object-oriented Programming http://en.wikipedia.org/wiki/Object-oriented_programming

[2] Swift Programming 101: Inheritance & Polymorphism http://www.iphonelife.com/blog/31369/swift-programming-101-inheritance-polymorphism

[3] Enumrations
https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Enumerations.html

[4] Protocols
https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Protocols.html#//apple_ref/doc/uid/TP40014097-CH25-XID_402

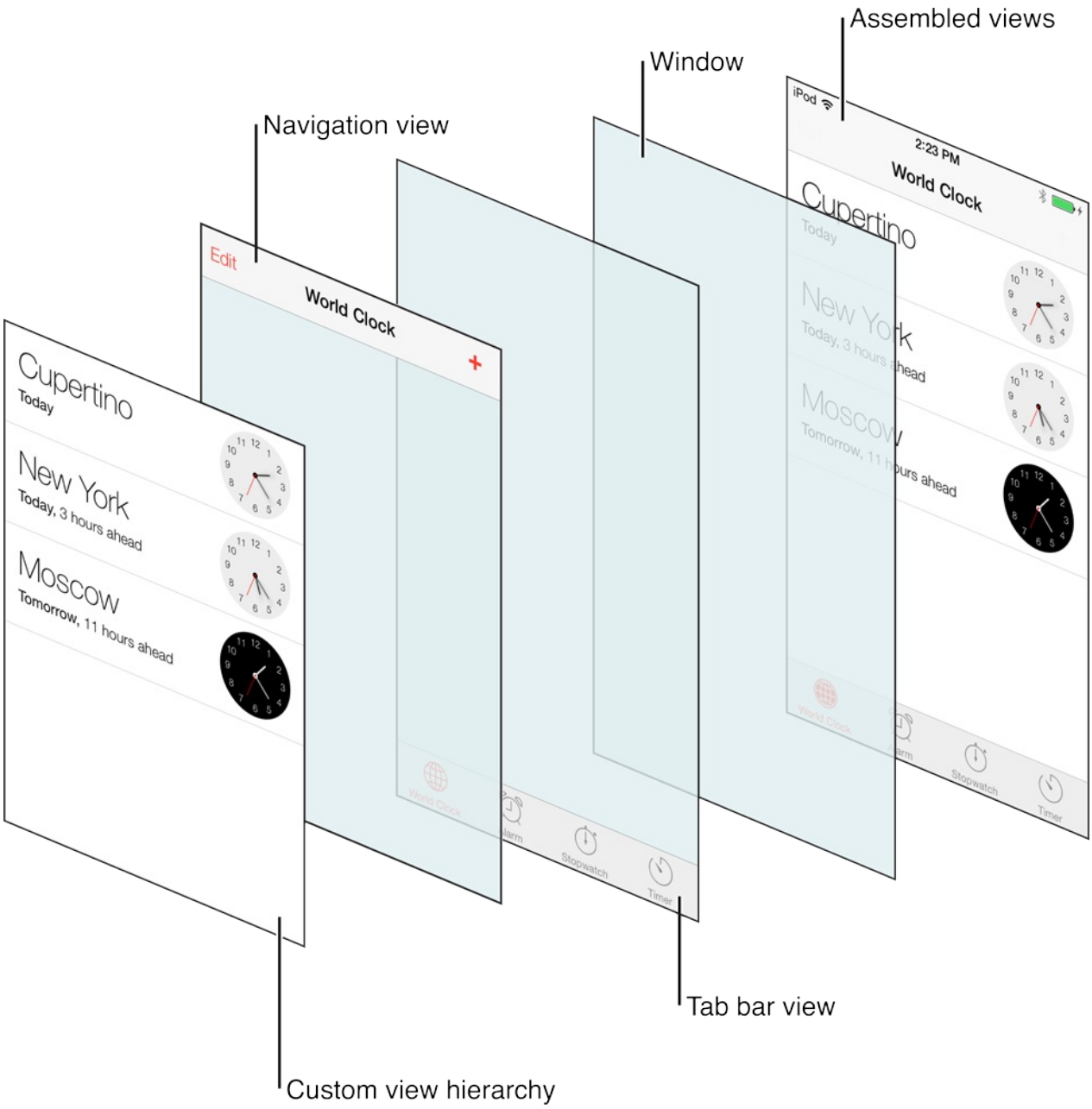
[5] Structures and Classes
https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ClassesAndStructures.html

[6] Protocols
https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Protocols.html

UIKit

Apple includes a variety of controls that serve very well in the creation of UI through Interface Builder. These controls are found in the Object Library within Xcode

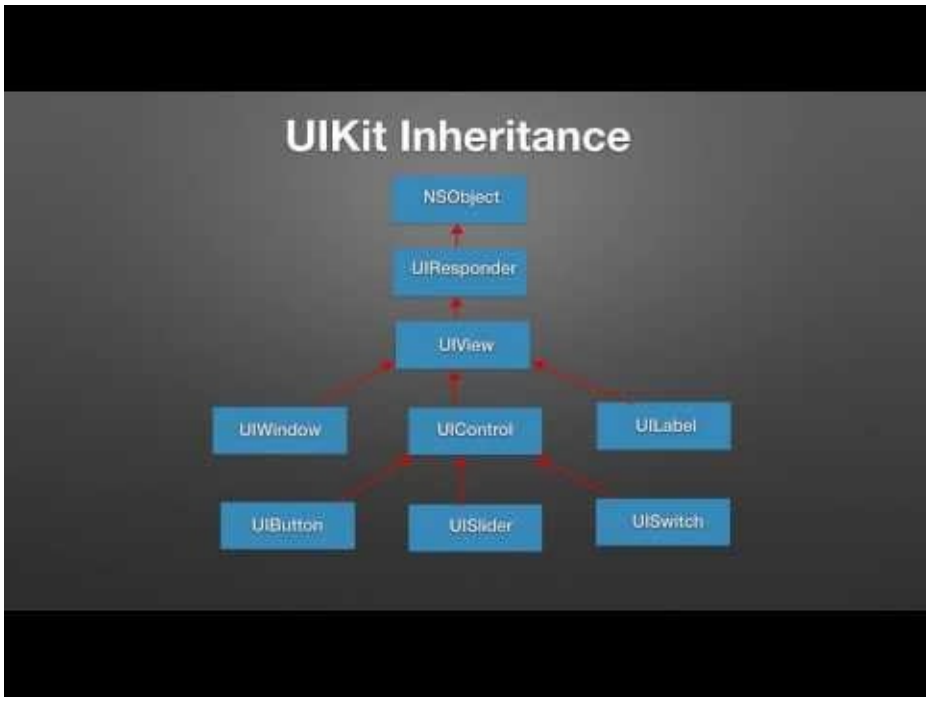
Overview



Apple provides excellent overview documentation on UIKit here

> Views are the building blocks for constructing your user interface. Rather than using one view to present your content, you are more likely to use several views, ranging from simple buttons and text labels to more complex views such as table views, picker views, and scroll views. [1]

[Video] Introduction to UIKit

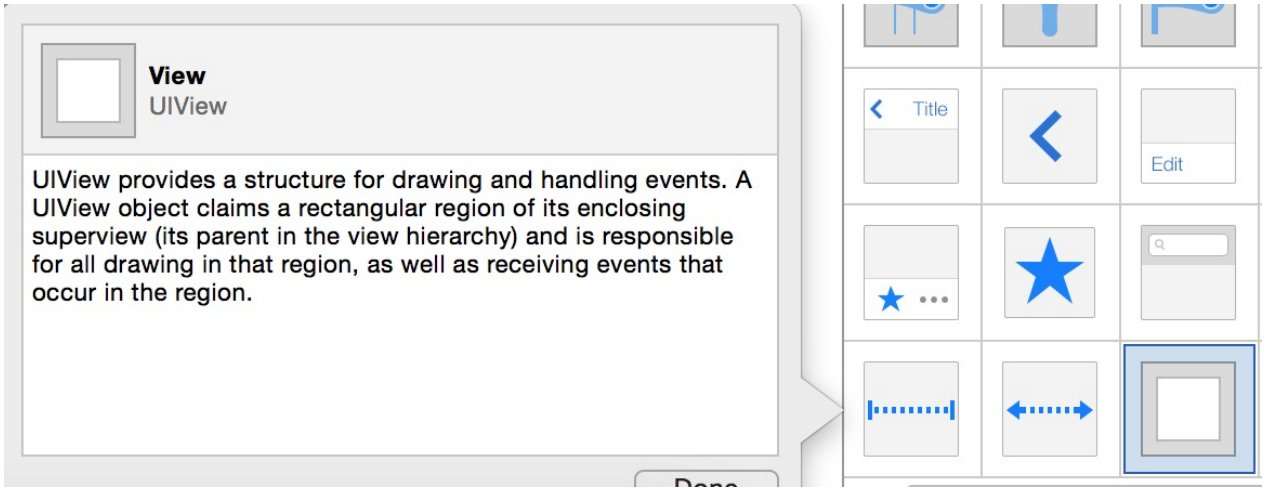


(http://www.youtube.com/watch?v=DFsENma-PAk

Common Controls

UIView

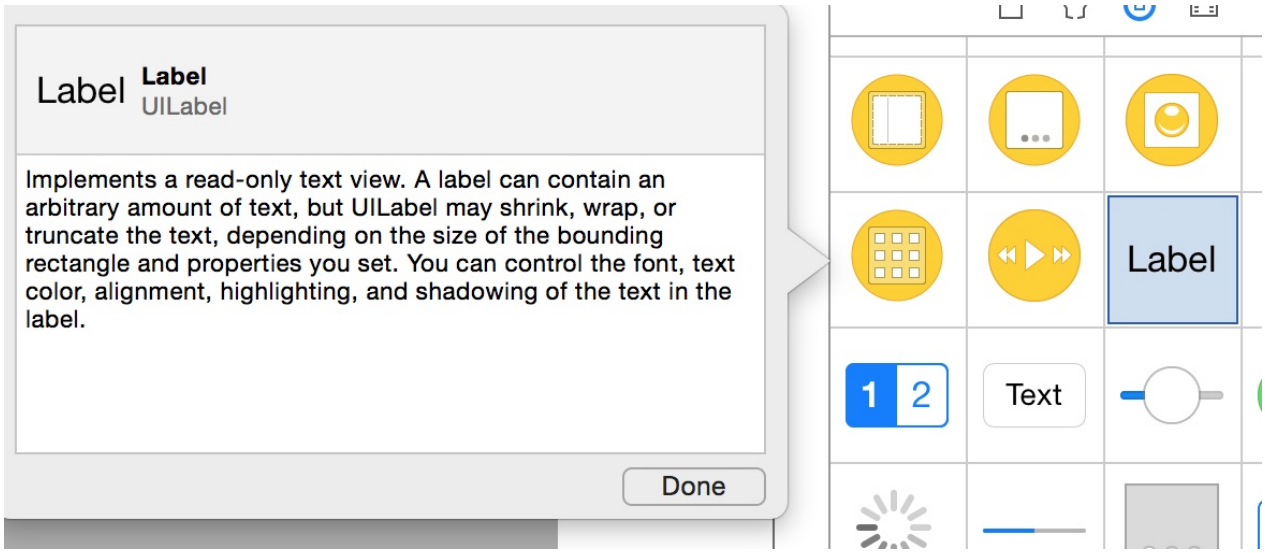
UIView is the basis for all views within iOS. Buttons, labels, tables are all subclasses of UIView.



> The UIView class defines a rectangular area on the screen and the interfaces for managing the content in that area.

Reference: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIView_Class/

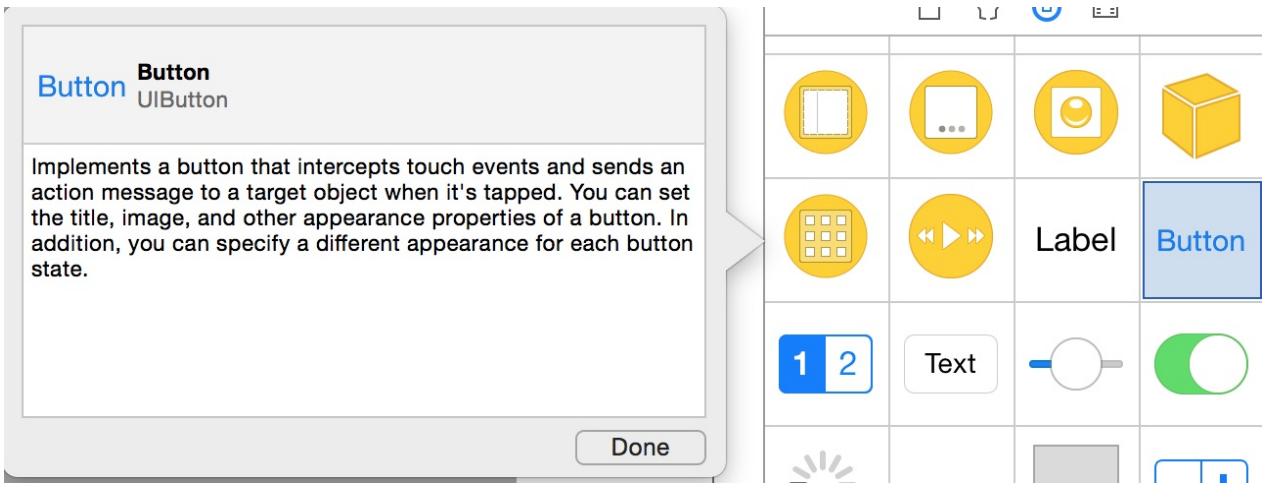
UILabel



> Use this class to draw one or multiple lines of static text, such as those you might use to identify other parts of your user interface.

Reference: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UILabel_Class/

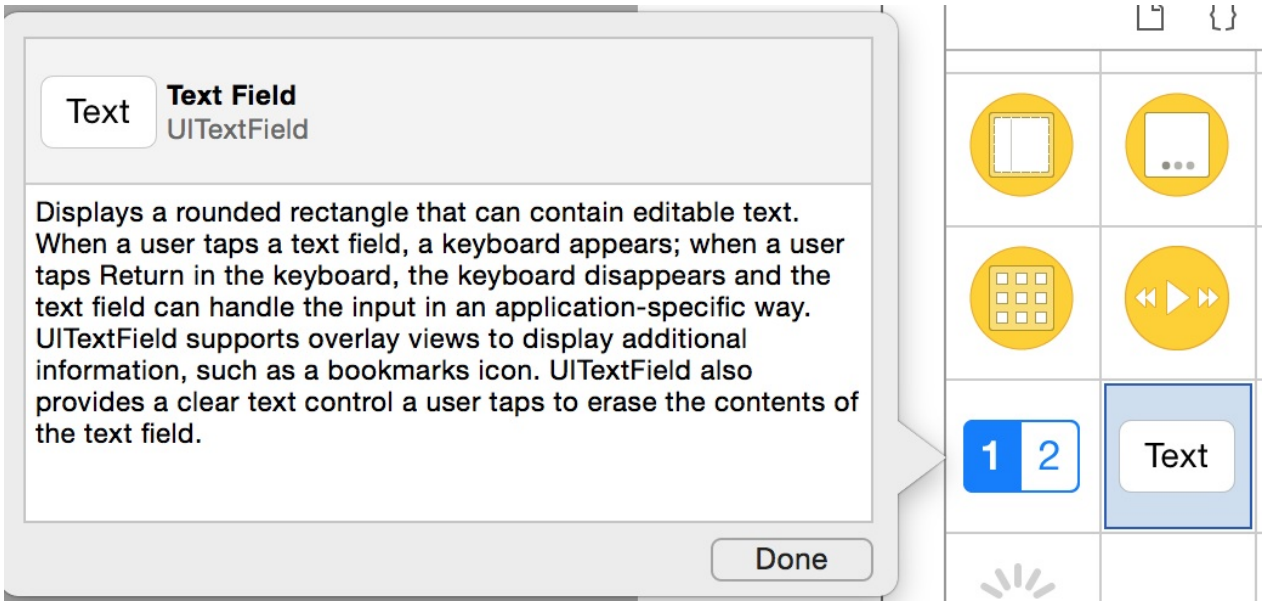
UIButton



> A button intercepts touch events and sends an action message to a target object when tapped.

Reference: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIButton_Class/index.html

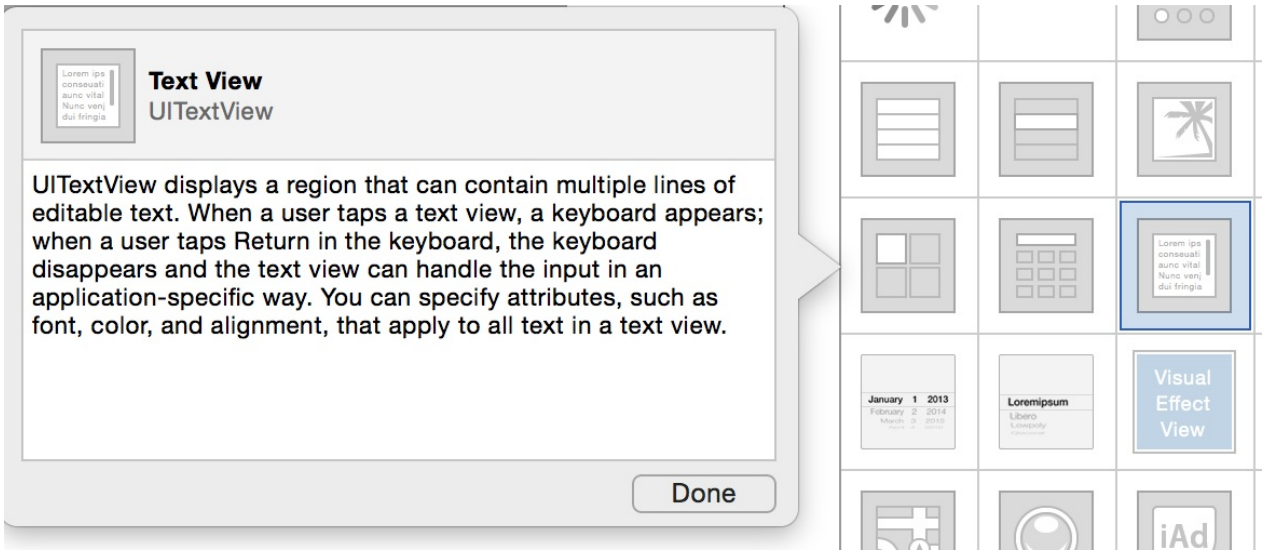
UITextField



> A UITextField object is a control that displays editable text and sends an action message to a target object when the user presses the return button

Reference: https://developer.apple.com/library/prerelease/ios/documentation/UIKit/Reference/UITextField_Class/index.html

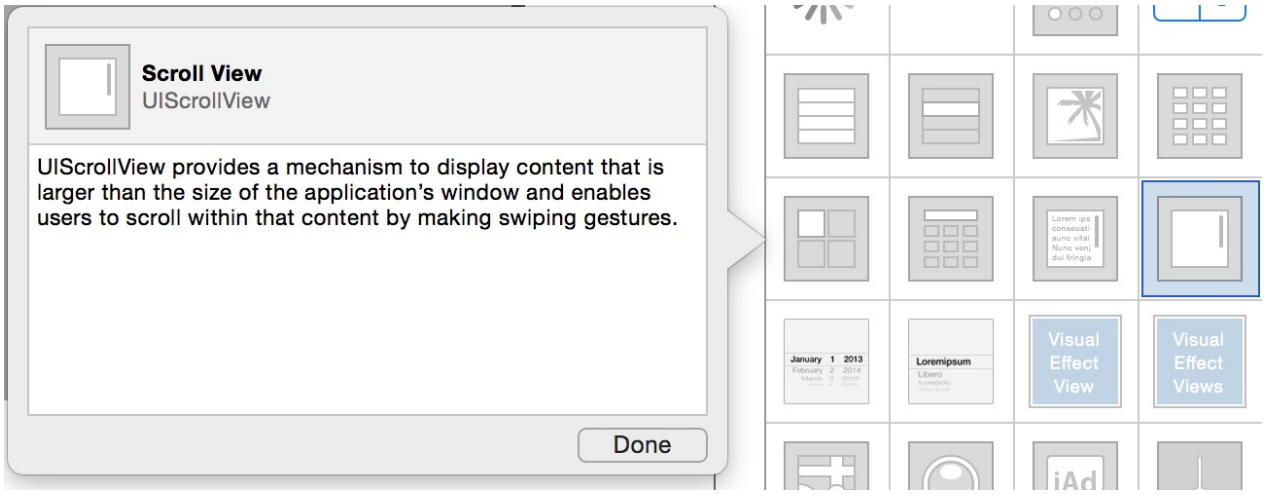
UITextView



> The UITextView class implements the behavior for a scrollable, multiline text region

Reference: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITextView_Class/index.html

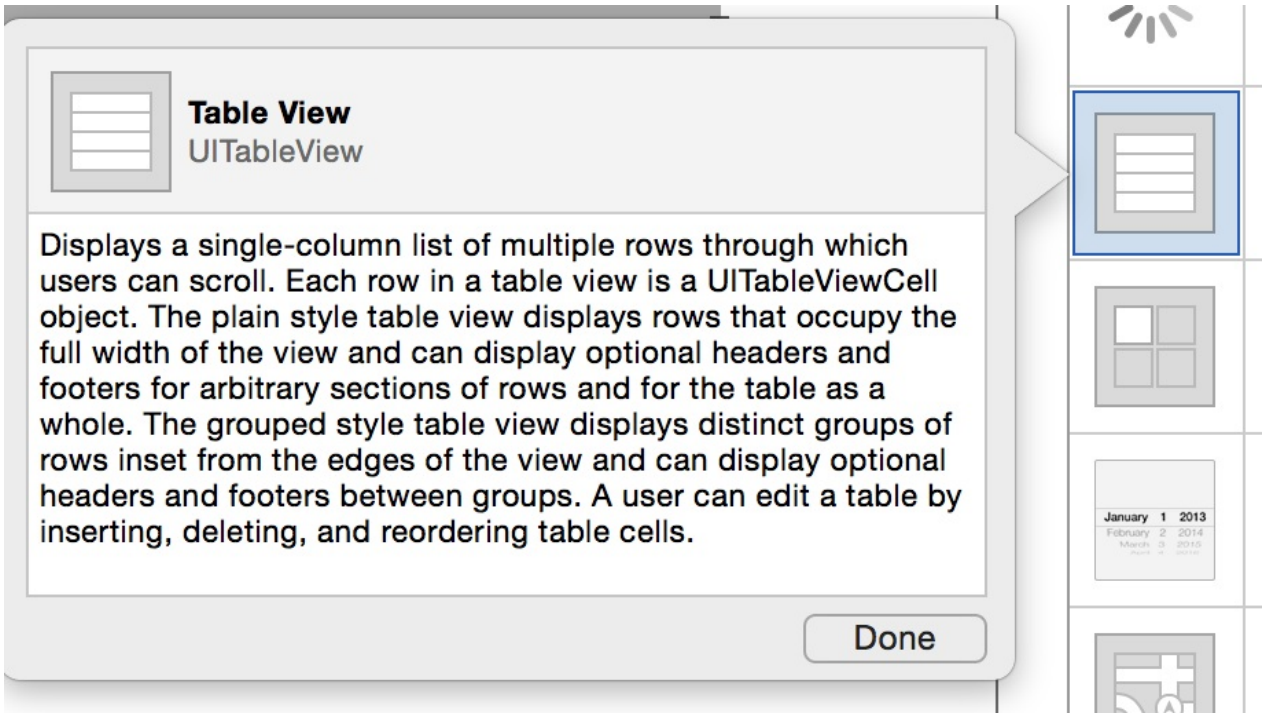
UIScrollView



> The UIScrollView ... enables users to scroll within that content by making swiping gestures, and to zoom in and back from portions of the content by making pinching gestures

Reference: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIScrollView_Class/index.html

UITableView

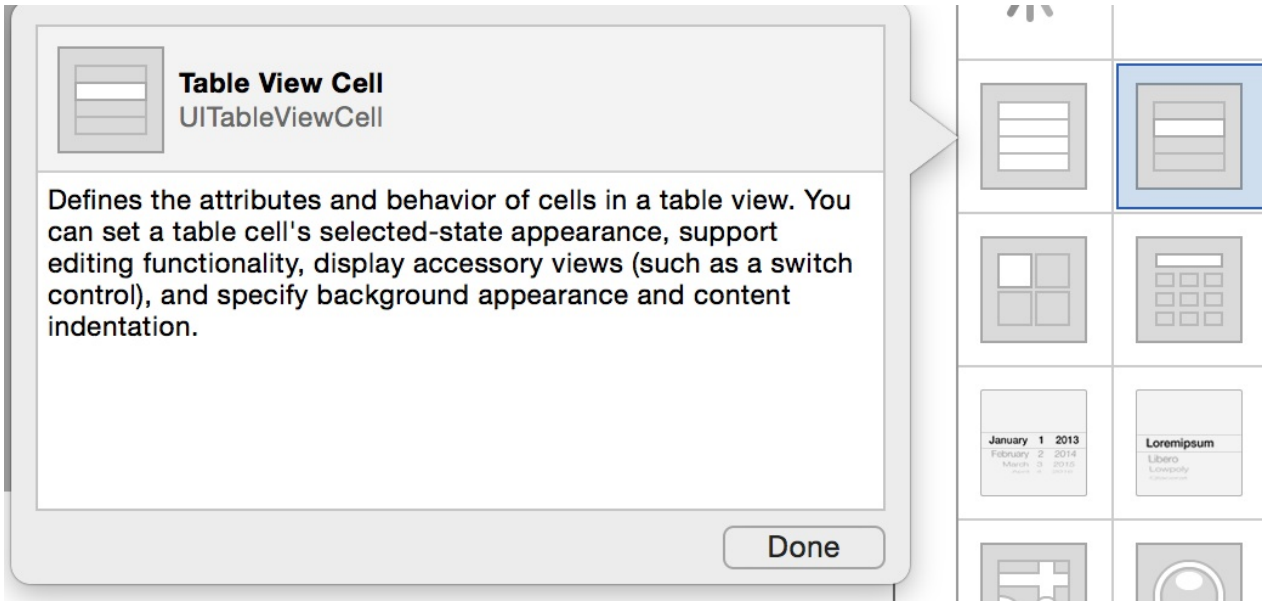


The UITableView allows for a vertical, scrolling list of cells to be displayed.

> A table view is made up of zero or more sections, each with its own rows. Sections are identified by their index number within the table view, and rows are identified by their index number within a section.

Reference: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableView_Class/index.html

UITableViewCell

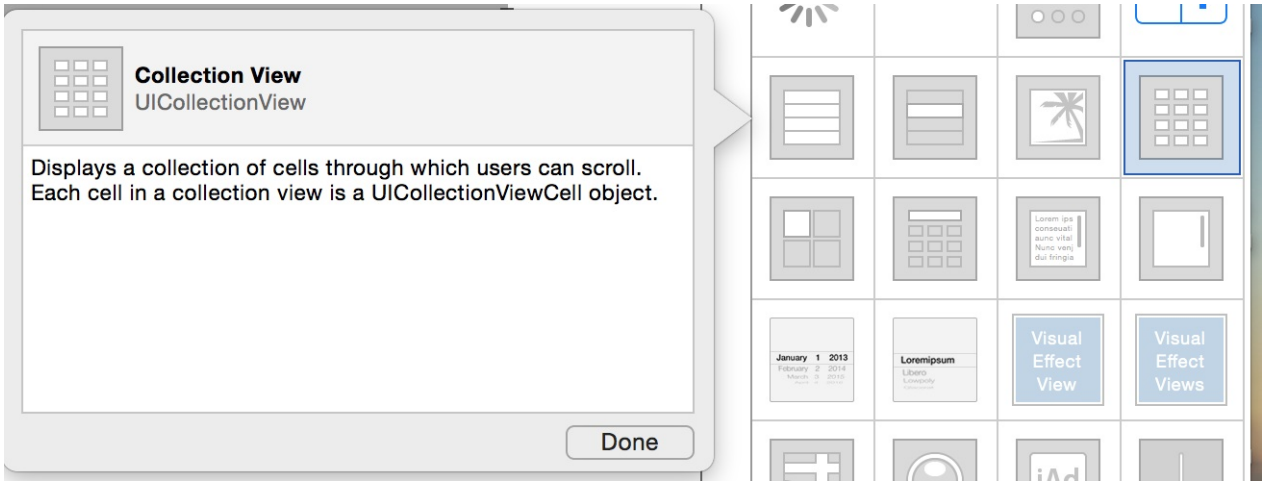


Manages content within UITableView.

> Includes properties and methods for setting and managing cell content and background (including text, images, and custom views), managing the cell selection and highlight state, managing accessory views, and initiating the editing of the cell contents

Reference: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITableViewCell_Class/

UICollectionView

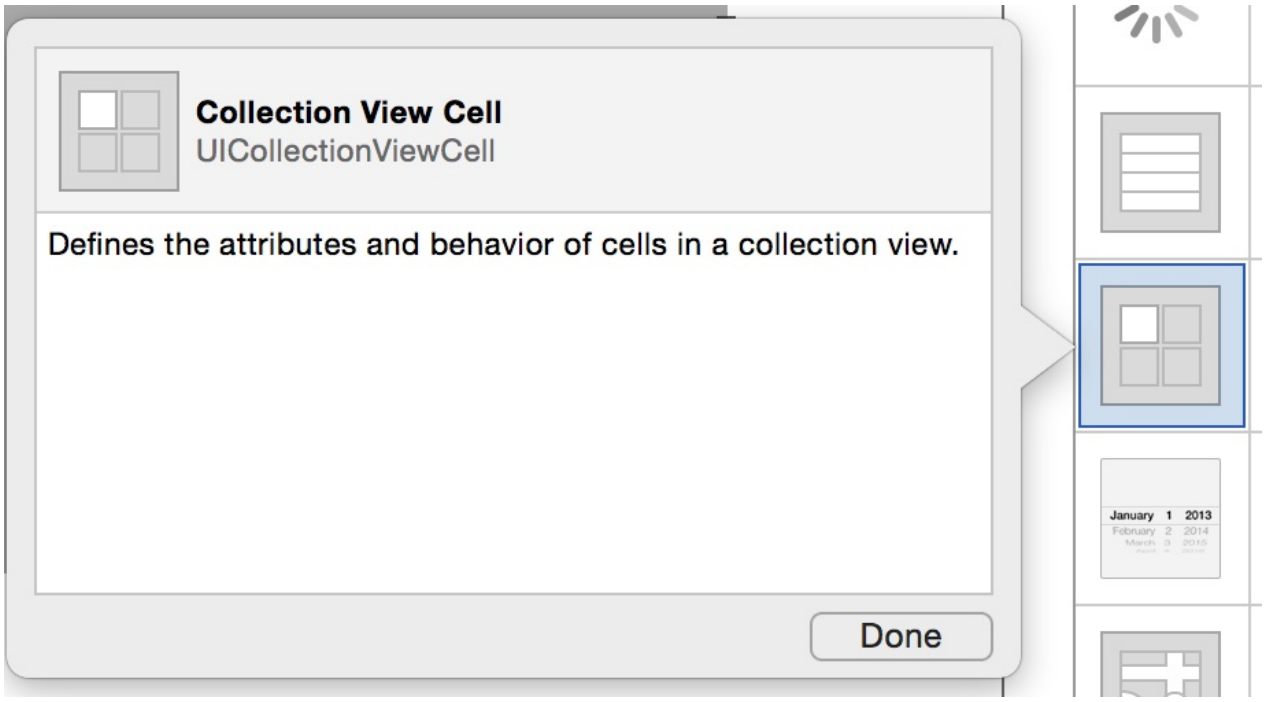


UICollectionViews are similar to UITableViews in that they display a series of cells. The extend the limitations of UITableView:

> Collection views support customizable layouts that can be used to implement multi-column grids, tiled layouts, circular layouts, and many more. You can even change the layout of a collection view dynamically

Reference: https://developer.apple.com/library/ios/documentation/UIKit/Reference/UICollectionView_class/index.html

UICollectionViewCell



> A UICollectionViewCell object presents the content for a single data item when that item is within the collection view's visible bounds.

Reference:
https://developer.apple.com/library/prerelease/ios/documentation/UIKit/Reference/UICollectionViewCell_class/index.html

---

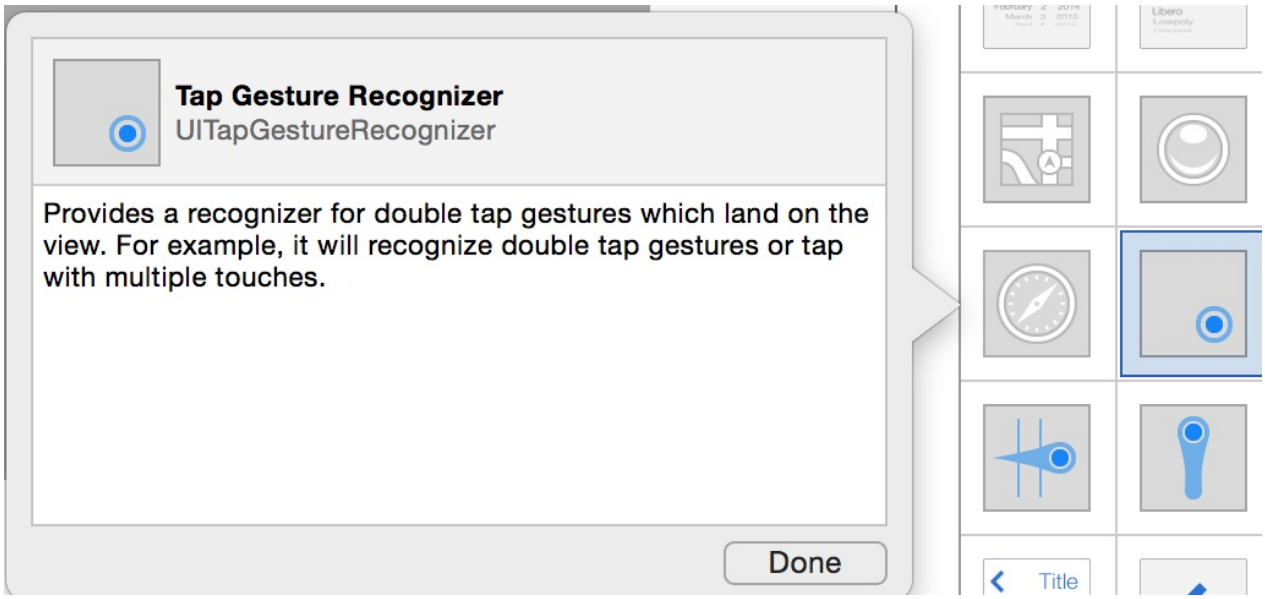
Common Gestures

Below are three of the most common gesture recognizers Apple provides in UIKit. They are all derived from a common base class of UIGestureRecognizer.

[Video] iOS Tutorial - UIGestureRecognizer

https://www.youtube.com/watch?v=KnfQRy6xOPk

UITapGestureRecognizer



> [UITapGestureRecognizer] looks for single or multiple taps. For the gesture to be recognized, the specified number of fingers must tap the view a specified number of times.

Reference:
https://developer.apple.com/library/ios/documentation/UIKit/Reference/UITapGestureRecognizer_Class/index.html

UIPanGestureRecognizer

> [UIPanGestureRecognizer] looks for panning (dragging) gestures. The user must be pressing one or more fingers on a view while they pan it.

Reference:
https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIPanGestureRecognizer_Class/index.html

UILongPressGestureRecognizer

> [UILongPressGestureRecognizer] looks for long-press gestures. The user must press one or more fingers on a view and hold them there for a minimum period of time before the action triggers.

Reference:
https://developer.apple.com/library/ios/documentation/UIKit/Reference/UILongPressGestureRecognizer_Class/index.html

---

References

[1] Apple - About Views https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/UIKitUICatalog/

[2] Apple UIKit Documentation https://developer.apple.com/library/ios/documentation/UIKit/Reference/

[3] Designing for iOS https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/

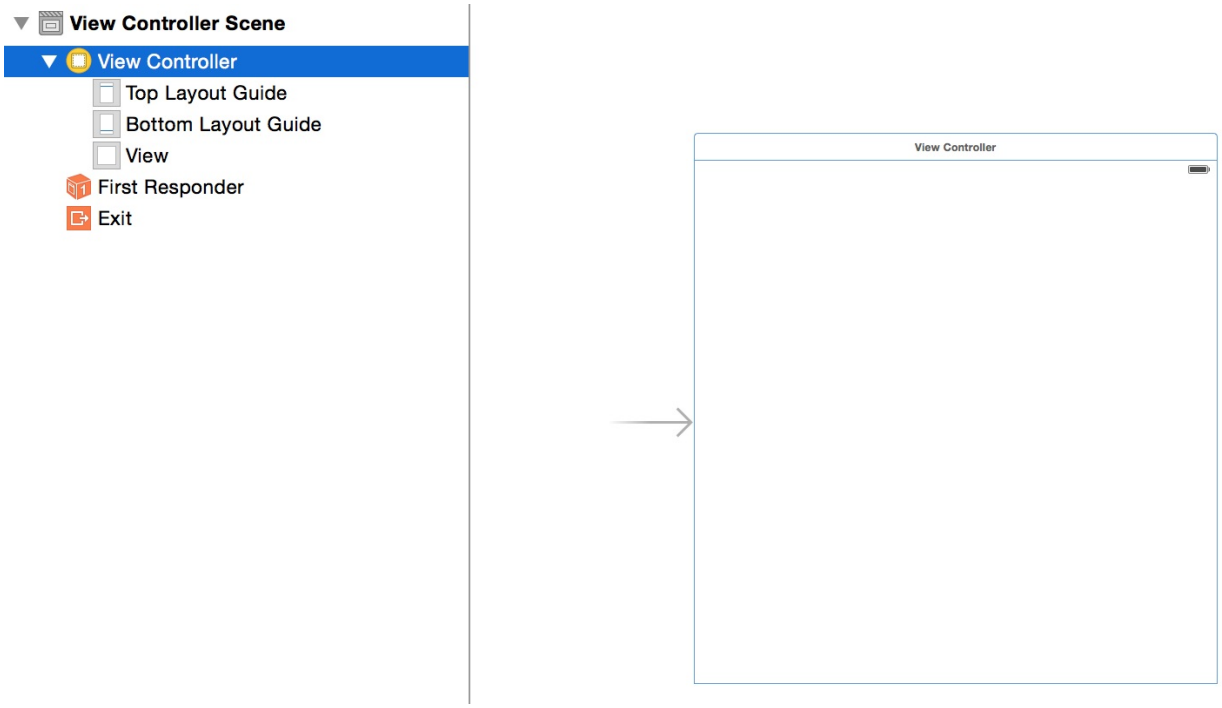
[4] Start Developing for iOS - User Interfaces
https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/DesigningaUserInterface.html

View Controllers

Apple includes a variety of View Controllers which organize your views and provide a controller class for delegates and other interactions.

UIViewController

UIViewController is the building block for user interfaces. You will typically subclass UIViewController in the development of your own view controllers.

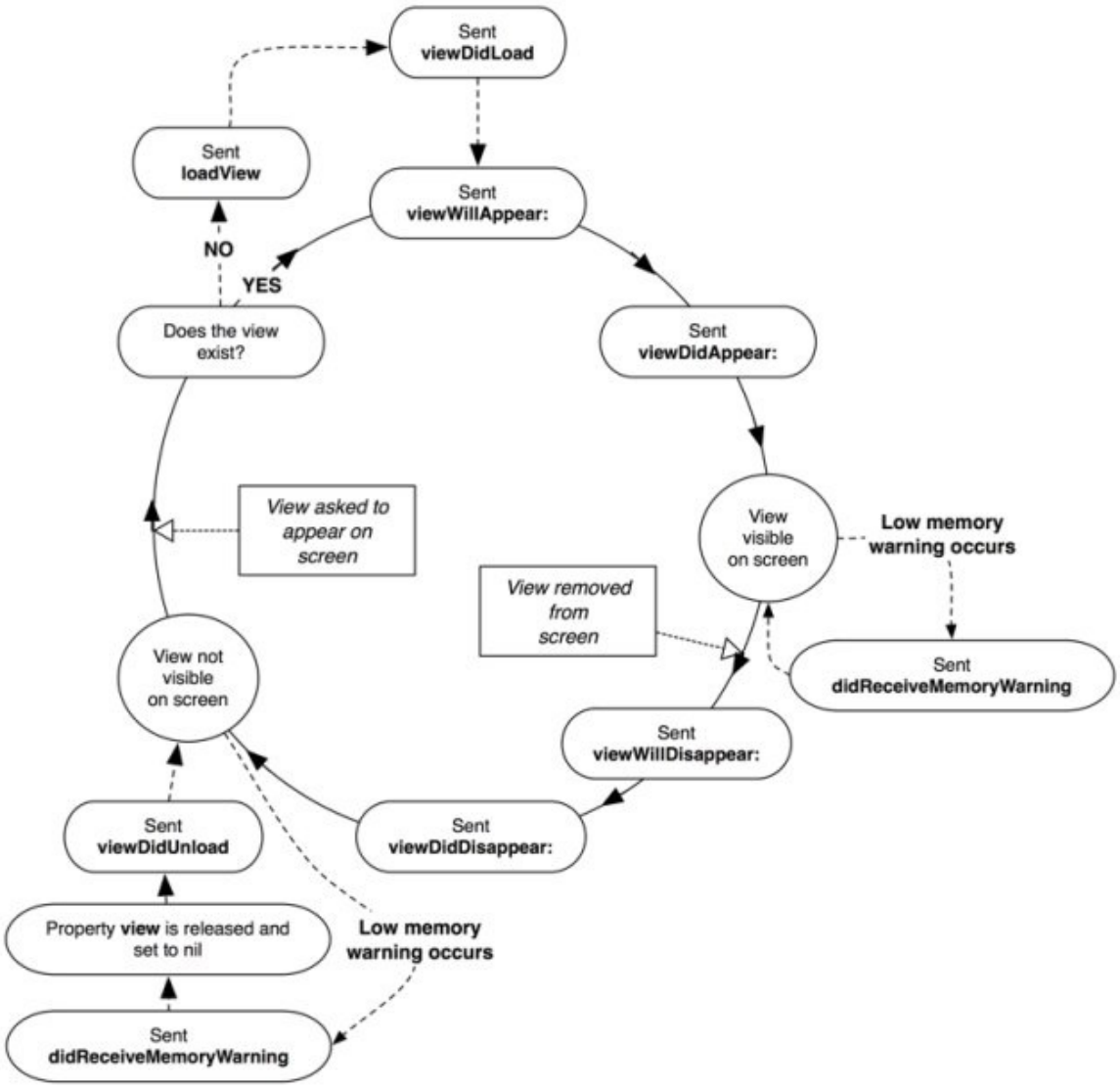


> The UIViewController class provides the fundamental view-management model for all iOS apps

https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIViewController_Class/index.html

View Controller Life Cycle

UIViewController provides a number of methods that you can override at various points of the view controller's life cycle.



Source: http://rdkw.wordpress.com/2013/02/24/ios-uiviewcontroller-lifecycle/

Common methods to overide

Depending on what you want to accomplish, the following methods can be overridden. These are all optional overrides.

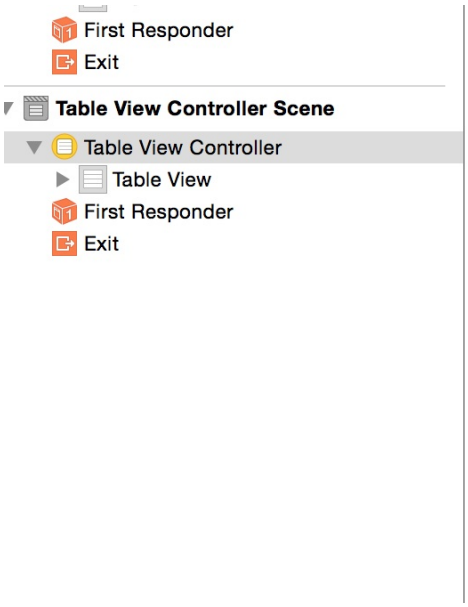
Method name	Common Use Case
viewDidLoad:	One-time setup, called when app has already loaded the XIB but has not started to display
viewWillAppear:	Called each time the view will begin to appear on screen. Setup data on screen such as labels.
viewDidAppear:	Called when the view has completed being displayed to the user. Use for custom animations on hidden views.
viewWillDisappear:	Called each time the view is about to begin disappearing. Use for more custom animations and for saving data to disk.
viewDidDisappear:	View has disappeared and view controller may be removed from memory.
didReceiveMemoryWarning	iOS will begin trying to reclaim memory. Release memory that can be recreated easily.

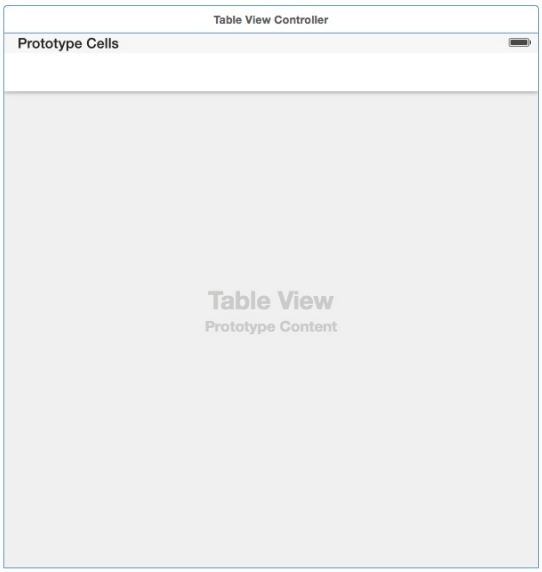
Derivatives

UIViewController is used as the base class for other common view controllers including **UITableViewController** and **UICollectionViewController**. These two additional classes will include other protocols that will need to be defined within

the implementation.

UITableViewController

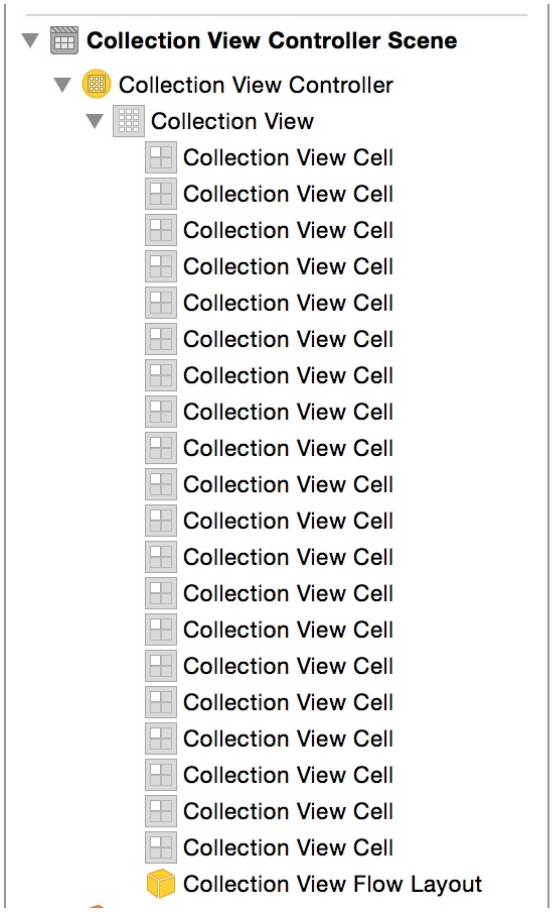


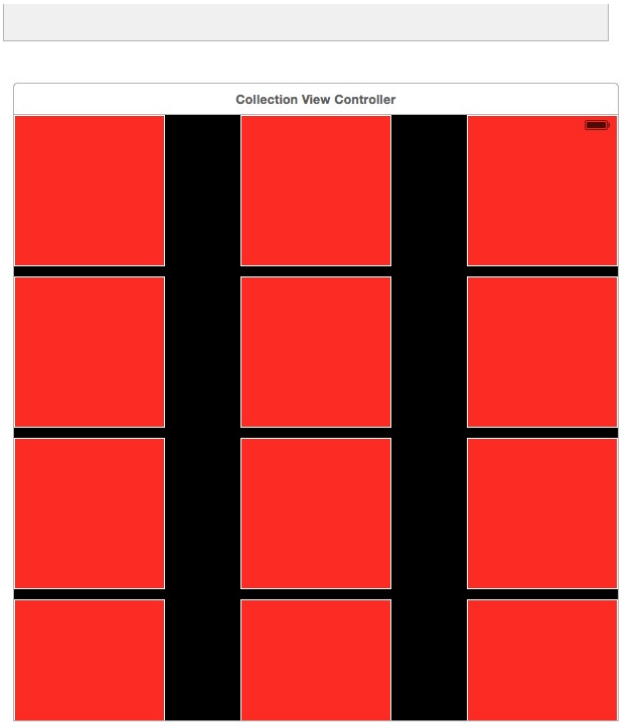


> The UITableViewController class creates a controller object that manages a table view.

https://developer.apple.com/library/prerelease/ios/documentation/UIKit/Reference/UITableViewController_Class/index.html

UICollectionViewController





> The UICollectionViewController class represents a view controller whose content consists of a collection view

https://developer.apple.com/library/prerelease/ios/documentation/UIKit/Reference/UICollectionViewController_clas/index.ht

ml

References

Apple UIKit Reference https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIKit_Framework/

View Controller Programming Guide
https://developer.apple.com/library/ios/featuredarticles/ViewControllerPGforiPhoneOS/CreatingCustomContainerViewControllers/CreatingCustomContainerViewControllers.html