

# CS 329E

# Elements of Mobile Computing

Fall 2017

University of Texas at Austin

Lecture 7

# Agenda

- Protocols
- Delegates
- Table View Controllers
- Navigation Controllers
- In-Class Exercise

# Protocols

# Protocols

Why do we care about protocols and delegates?

They are used extensively in iOS development.

They are used in a new programming paradigm:

Protocol-Oriented Programming  
(as opposed to Object-Oriented Programming)

# Protocols

What is a protocol?

- A *blueprint* of methods, properties and other requirements that suit a particular task or piece of functionality.
- Similar to interfaces in Java, but more powerful and flexible.
- Can be adopted by a class, structure or enumeration to provide an actual implementation of the defined functionality.
- Any type (class, struct, enum) that satisfies the requirements of a protocol is said to *conform* to that protocol.

# Protocols

What is a protocol? (continued)

- Frequently, the protocol has no implementation.
- But you can provide a default implementation by *extending* a protocol.
- A way to have *fake* single or multiple inheritance.

# Protocols

What is a protocol? (continued)

- A group of related properties and methods that can be implemented in *any* class. As such, they are independent of any class.
- When you define a protocol you identify what is *required* and what is *optional*.
  - Anything *required* must be implemented.
- Just like classes, protocols can inherit from other protocols.

# Protocols

- It's also a way to define a set of methods that are implemented in unrelated classes
  - Unlike inheritance, where there is a clear relationship
- All that is required is that you indicate you conform to the protocol and implement all the required methods
- Example: both an Employee and an (unrelated) Automobile class could implement the same protocol.



# Protocols

Protocols are declared using the **protocol** keyword.

When defining a protocol, we provide:

- The signatures for any **methods** that the protocol should implement.
- The types of any **properties** that should be implemented along with whether they should implement both get and set methods or only get.
- Unless specified otherwise, everything is required.

# Protocols

Example:

```
protocol Bird {  
    var name: String { get }  
    var canFly: Bool { get }  
    func printBird()  
}
```

```
class SillyBird : Bird {  
  
}
```

# Protocols

Example (continued):

```
// This class conforms to the Bird protocol.
class SillyBird : Bird {
    // class-specific property.
    var prop1: Int = 0

    // class-specific method.
    func printMe() {
        print("SillyBird: \ \(prop1)")
    }

    // Protocol implementations.
    var name: String {
        get { return "Silly Bird" }
    }
    var canFly: Bool {
        get { return true }
    }
    func printBird() {
        print("SillyBird: printBird: name: \ \(name), canFly: \ \(canFly)")
    }
}
```

# Protocols

Real-world iOS protocol example:

- *UITableViewDataSource*
  - Defines the set of methods that are called by the framework to get something to display in a table view.

# Protocols

```
protocol UITableViewDataSource : NSObjectProtocol {  
    . . . . .  
  
    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int  
  
    // Row display. Implementers should *always* try to reuse cells by setting each cell's  
    // reuseIdentifier and querying for available reusable cells with dequeueReusableViewWithReuseIdentifier:  
    // Cell gets various attributes set automatically based on table (separators) and data source  
    // (accessory views, editing controls)  
  
    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->  
    UITableViewCell  
  
    // Editing  
  
    // Individual rows can opt out of having the -editing property set for them. If not implemented, all  
    // rows are assumed to be editable.  
    optional func tableView(tableView: UITableView, canEditRowAtIndexPath indexPath: NSIndexPath) -> Bool  
  
    // Moving/reordering  
  
    // Allows the reorder accessory view to optionally be shown for a particular row. By default, the  
    // reorder control will be shown only if the datasource implements -  
    // tableView:moveRowAtIndexPath:toIndexPath:  
    optional func tableView(tableView: UITableView, canMoveRowAtIndexPath indexPath: NSIndexPath) -> Bool  
  
    . . . . .  
}
```

# Protocols

Benefits of protocols:

- It enables loose coupling of components.
- Using protocols instead of subclassing gives developers much more leeway when it comes to organizing their application's code.
- Allows you to very easily add a protocol to a type (class, struct, enum) thus giving it that capability without concern for something like a class hierarchy.

# Delegates

# Delegates

Protocols and delegates are closely associated, in that, in order for protocols to be useful in a lot of cases you need at least one delegate to conform to the protocol.

What is a delegate?

- A pointer to *some object* that has implemented the protocol's methods (conformed to the protocol).
- The *some object* means we don't really know, or care, specifically what kind of object the delegate is referring to, only that the *required* methods/properties defined in the protocol are implemented in that object.



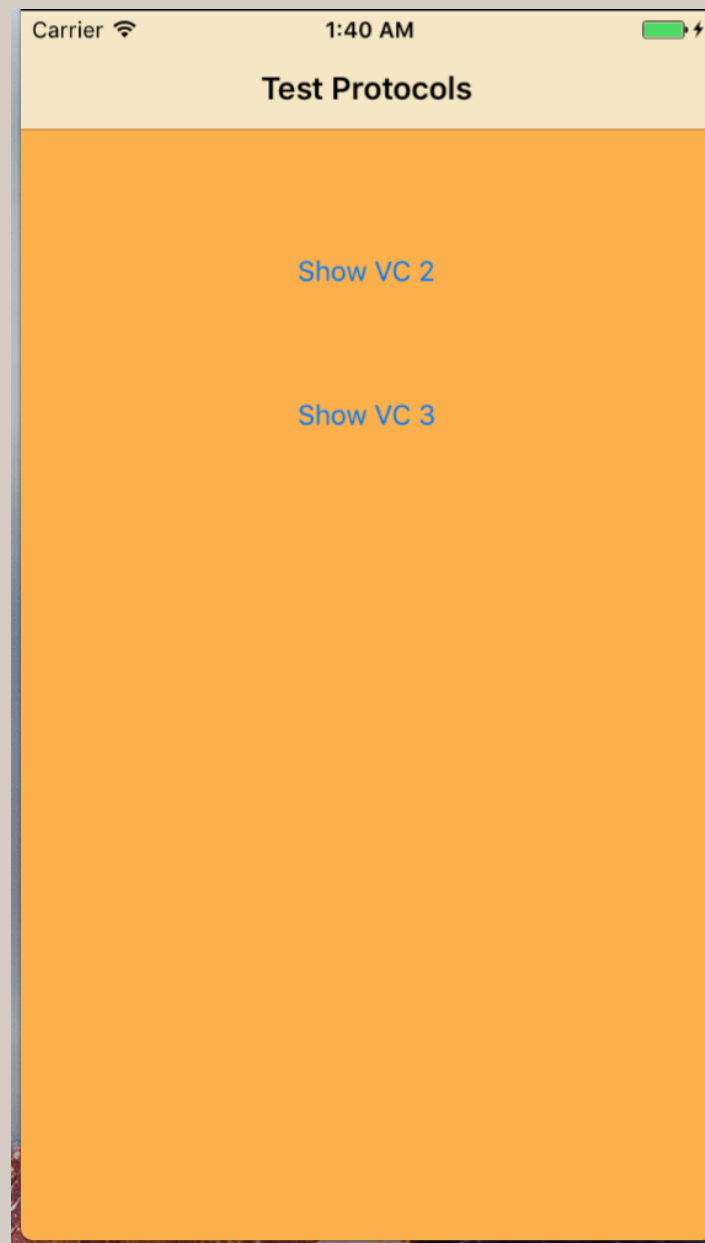
# Delegates

What is delegation? What are delegates?

- Delegation is a simple and powerful pattern in which one object in a program acts on behalf of, or in coordination with, another object.
- The *delegating* object keeps a reference to another object - the *delegate* - and at the appropriate time makes use of the delegates implementation.
- The main value of delegation is that it allows you to easily customize the behavior of several objects (delegates) in one central object (delegator).
- Used extensively in normal iOS programming.

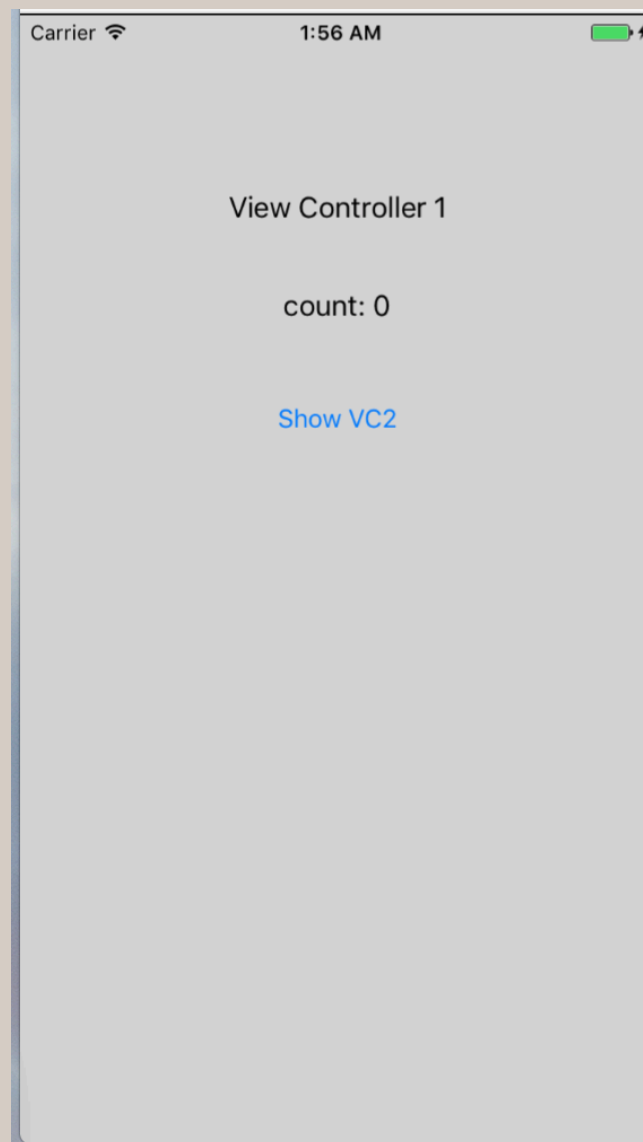
# Delegates

Demo: TestProtocols



# Delegates

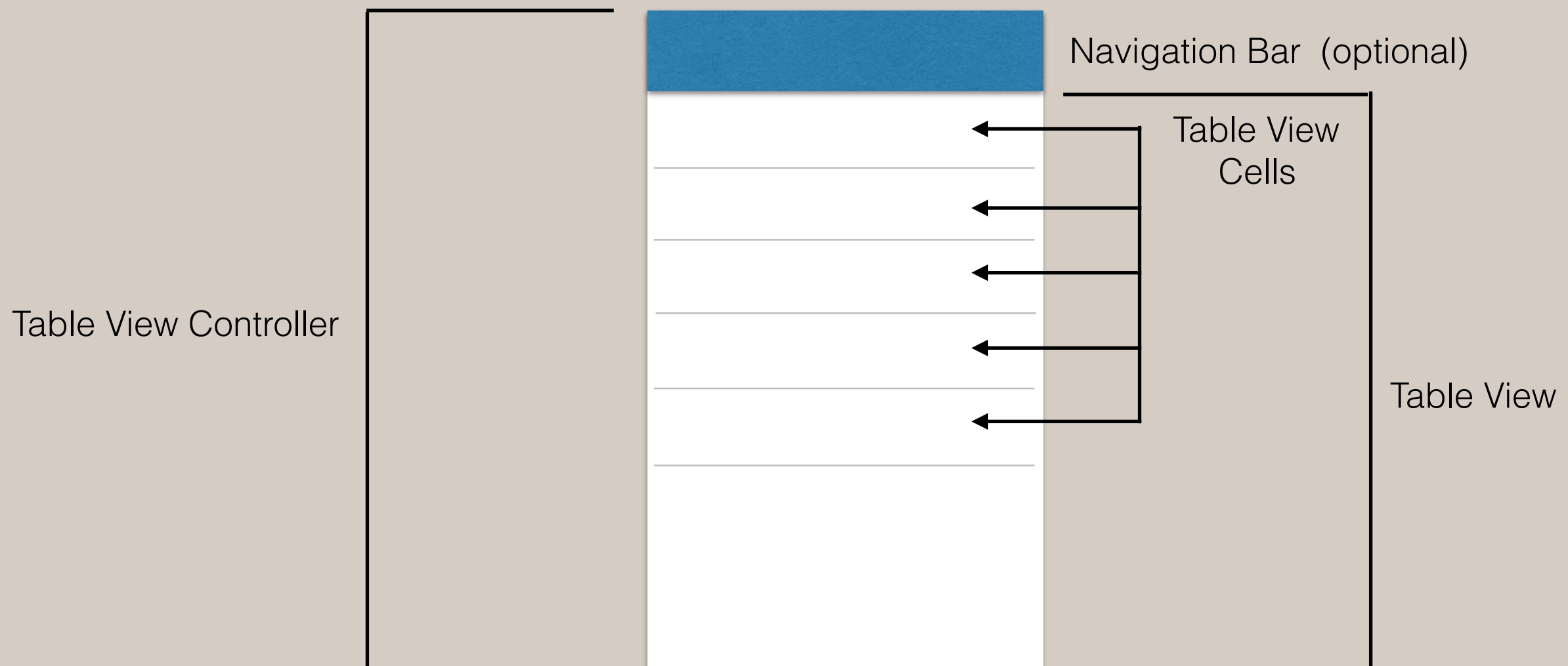
Demo: TestProtocolsAndDelegates



# Table View Controllers

# Table View Controllers

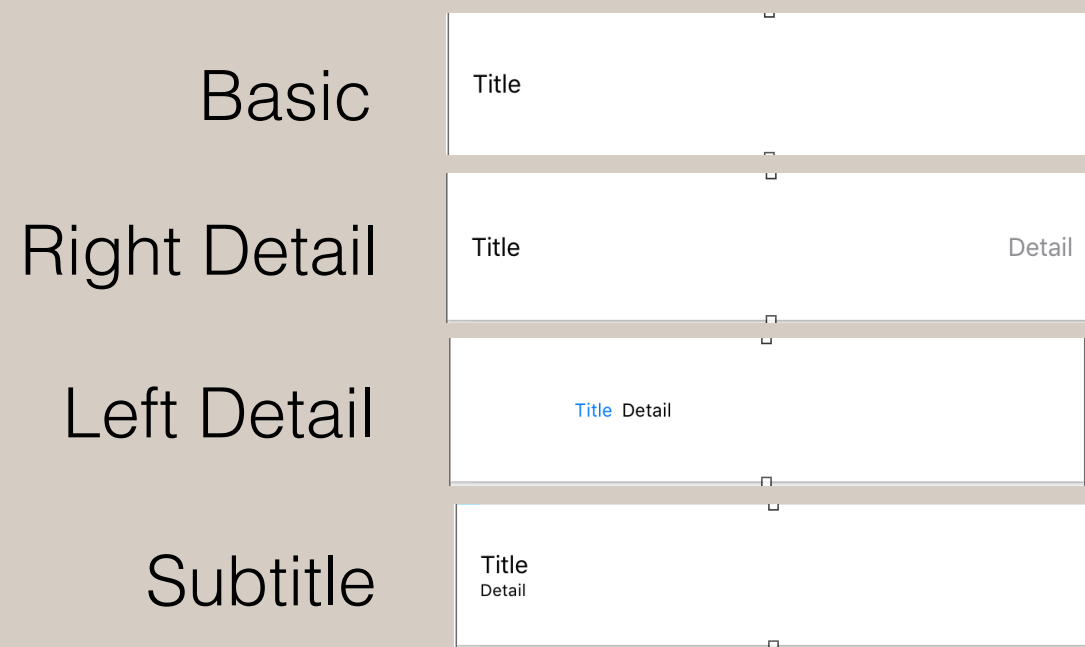
- A UITableViewController contains a UITableView.
- A UITableView contains *at least one* UITableViewCell.
- A UITableViewCell contains UI elements - Labels, Buttons, etc.



# Table View Controllers

A UITableViewCell has a *style* property.

The pre-defined table view cell styles are:



There is also the *custom* table view cell style.

# Table View Controllers

- A UITableViewCell contains UI elements - Labels, Buttons, etc.
- The outlet names for the text in the standard cell styles:
  - “Title” -> titleLabel
  - “Detail” -> detailTextLabel
- The UILabel outlet names for the 4 standard cell styles:
  - Basic - titleLabel
  - Right Detail - titleLabel, detailTextLabel
  - Left Detail - titleLabel, detailTextLabel
  - Subtitle - titleLabel, detailTextLabel

# Table View Controllers

So, how does data appear in a Table View?

By creating a custom UITableViewController or UITableView derived class and implementing the necessary protocol methods as defined in the *UITableViewDataSource* protocol.

Most importantly:

- numberOfSections
- numberOfRowsInSection
- cellForRowAtIndexPath



# Table View Controllers

UITableViewController *conforms* to a number of protocols:

- The two most important in our case:
  - UITableViewDataSource - methods used to get data into the table view
  - UITableViewDelegate - methods used to handle actions of the table view

```
class UITableViewController : UIViewController, UITableViewDelegate,
                                NSObjectProtocol, UIScrollViewDelegate,
                                UITableViewDataSource {

    init(style: UITableViewStyle)
    init!(nibName nibNameOrNil: String!, bundle nibBundleOrNil: NSBundle!)
    init!(coder aDecoder: NSCoder!)

    var tableView: UITableView!
    @availability(iOS, introduced=3.2)
    var clearsSelectionOnViewWillAppear: Bool // defaults to YES. If YES, any
                                                // selection is cleared in
                                                // viewWillAppear:

    @availability(iOS, introduced=6.0)
    var refreshControl: UIRefreshControl?
}
```

# Table View Controllers

```
protocol UITableViewDataSource : NSObjectProtocol {
    . . . . .

    func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int

    // Row display. Implementers should *always* try to reuse cells by setting each cell's
    // reuseIdentifier and querying for available reusable cells with dequeueReusableViewWithIdentifier:
    // Cell gets various attributes set automatically based on table (separators) and data source
    // (accessory views, editing controls)

    func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) ->
    UITableViewCell

    // Editing

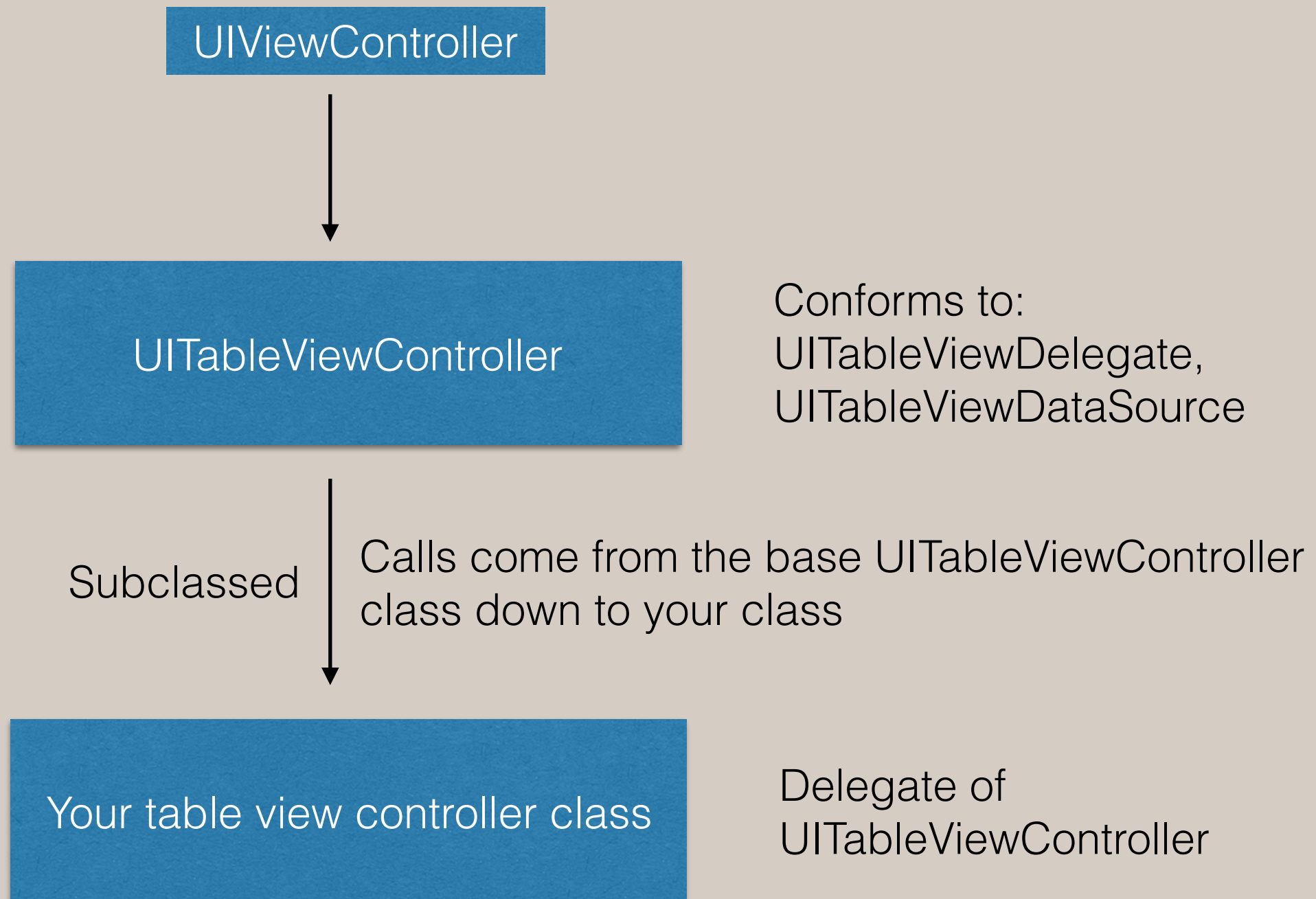
    // Individual rows can opt out of having the -editing property set for them. If not implemented, all
    // rows are assumed to be editable.
    optional func tableView(tableView: UITableView, canEditRowAtIndexPath indexPath: NSIndexPath) -> Bool

    // Moving/reordering

    // Allows the reorder accessory view to optionally be shown for a particular row. By default, the
    // reorder control will be shown only if the datasource implements -
    // tableView:moveRowAtIndexPath:toIndexPath:
    optional func tableView(tableView: UITableView, canMoveRowAtIndexPath indexPath: NSIndexPath) -> Bool

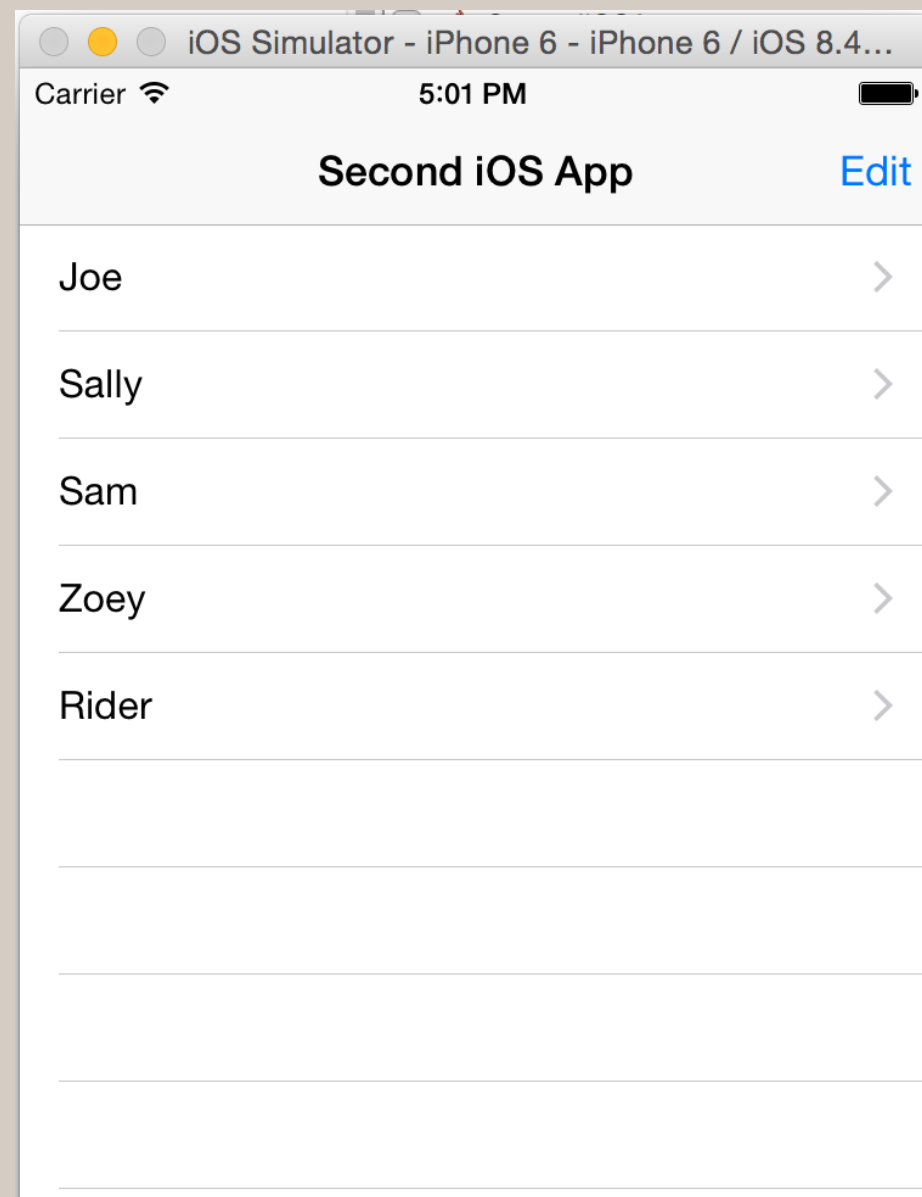
    . . . . .
}
```

# Table View Controllers



# Table View Controllers

An app with a Table View Controller using the *Basic* predefined table view cell style:



# Table View Controllers

Basic use:

- Derive a new class from UITableViewController.
- Code generated will include method signatures for all required methods, and some commented out optional methods.
- Minimally, you fill in the necessary code in the required methods and data appears in the table.
- Two of the more important early decisions are:
  - Whether to make use of one of the base table view cell styles or create a custom cell layout.
  - The structure of the app's data model.

# Navigation Controllers

# Navigation Controllers

What are Navigation Controllers?

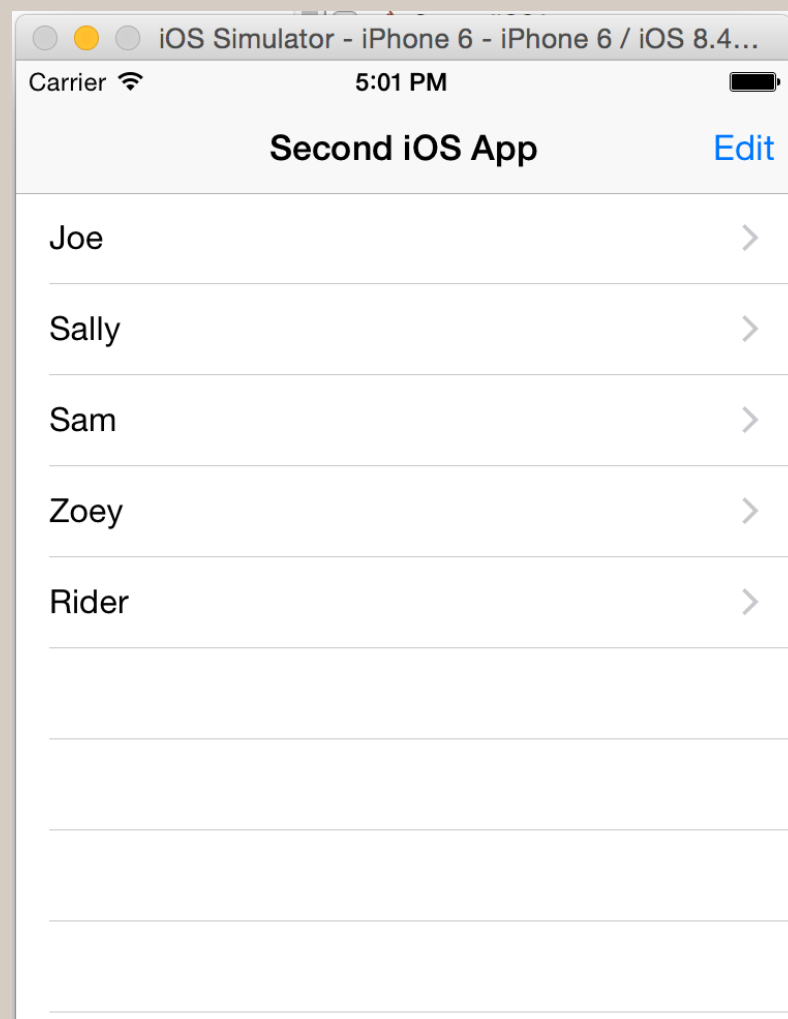
- Container controllers that contain some number of view controllers.
- The child view controllers are referenced in an internal array that is a list of view controllers that have been *pushed* onto the view controller stack - for display to the user; and *popped* from the view controller stack - for removal and display of the prior view controller.

# Navigation Controllers

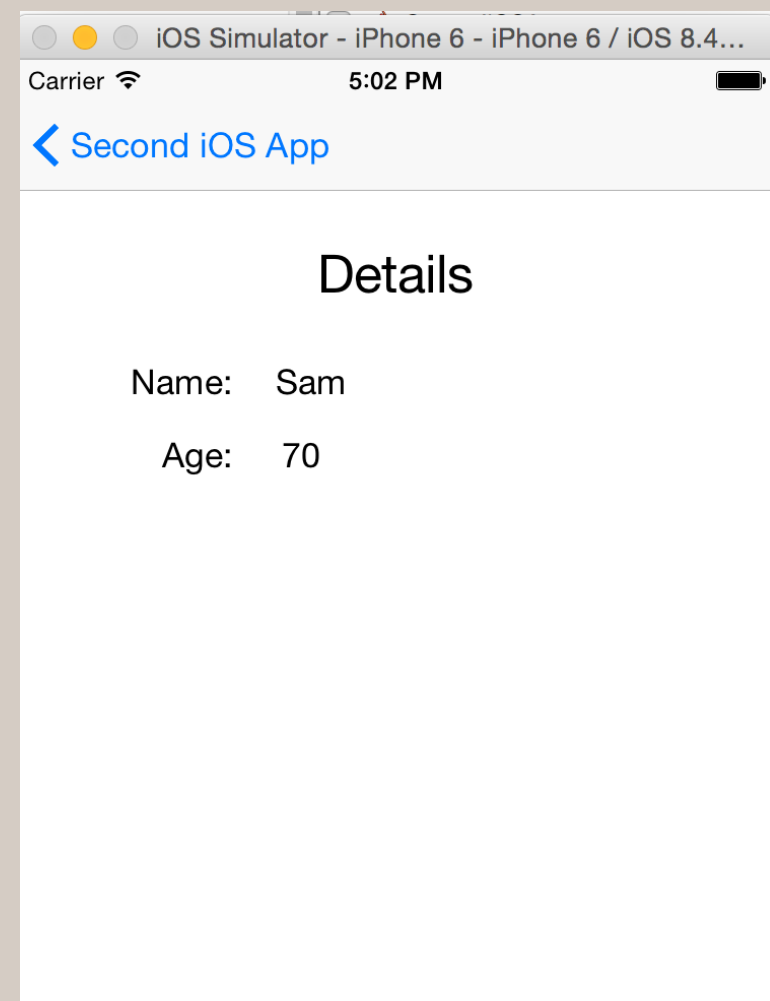
A typical app with a Navigation Controller:

- In this case, the detail view controller was *pushed* onto the view controller stack for display when the user touches one of the table rows

Table view controller



Detail view controller



*Popped*



*Pushed*





# Navigation Controllers

How to set the navigation bar title:

In viewDidLoad: (either one works)

```
self.title = "Add Person"
```

```
self.navigationItem.title = "Add Person"
```

One way to set the back button text:

In viewDidLoad:

```
let barButton = UIBarButtonItem()
```

```
barButton.title = "Back"
```

```
navigationController!.navigationBar.topItem!.backBarButtonItem = barButton
```

Note: Be careful about the combined width of the title and back button text.

# In-Class Exercise

# In-Class Exercise

Create an iOS app that:

- Includes a Table view controller embedded in a Navigation controller
- Segues to a simple view controller when a table view cell is touched
- Each table view cell will be populated with data from an element of a data model