Protocols and Delegates

Dr. Sarah Abraham

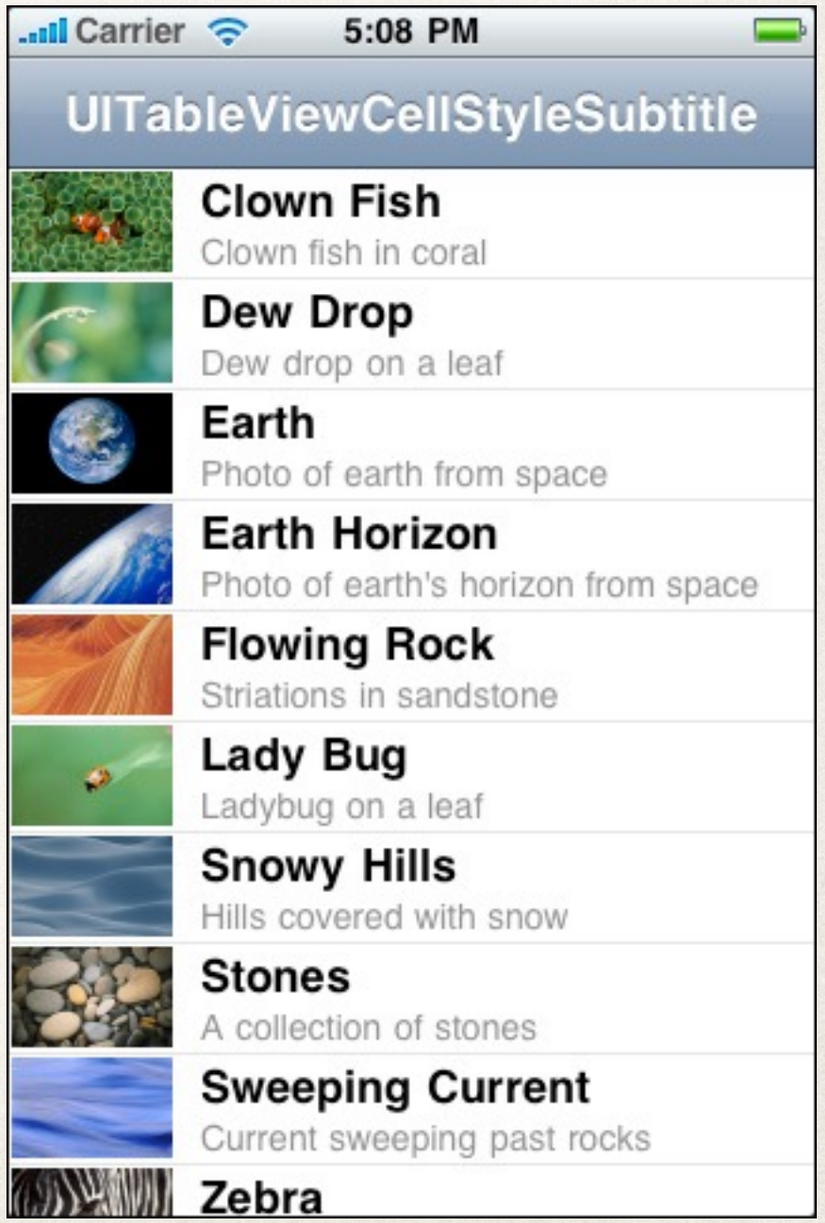
University of Texas at Austin
CS329e
Spring 2020

Protocols

- Group of related properties and methods that can be implemented by *any* class
- Independent of any class
  - Known in other languages as interfaces
  - Abstract with no default implementation
- Used throughout iOS development

Example: Table View

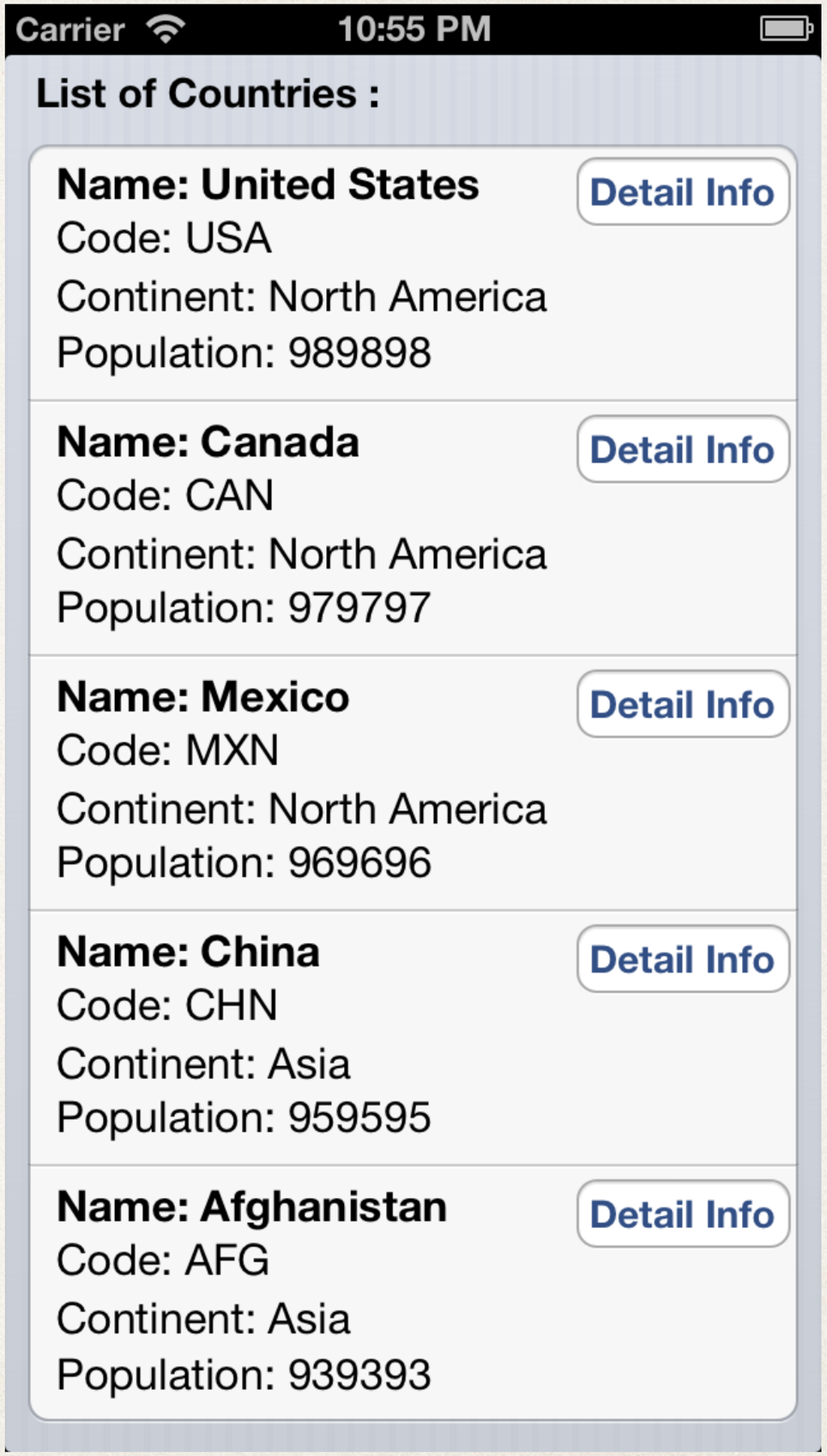
- A table view displays tabular information
  - View only — no data management
- Must talk to a *data source* to know what to display in its table
- Data source must respond to table view's messages
- Protocol declares methods expected (or required) for this interaction between table and data source



UITableViewDataSource Protocol

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

    func tableView(UITableView, numberOfRowsInSection: Int) -> Int
     /* Data source returns the number of rows in a given section of table
  view */

    func tableView(UITableView, cellForRowAt: IndexPath) -> UITableViewCell
    /* Data source returns cell to insert in particular location of table
    view */

    optional func tableView(UITableView, canEditRowAt: IndexPath) -> Bool
    /* Optional: Data source determines whether row is editable */

    optional func tableView(UITableView, canMoveRowAt: IndexPath) -> Bool
    /* Optional: Data source determines whether row is movable */


. . . . .

}
```

- UITableViewDataSource provides information to TableView about:
  - Number of cells
  - Content of the cells
  - Optional editing functionality



Adopting a Protocol

- Classes can adopt protocols by listing them *after* their super class:

```
class MyViewController: UIViewController,
UITextFieldDelegate {

    //class specification here

}
```

- Where have you seen this?

Using a Protocol

- Class must *conform* to the protocol
  - To conform to a protocol, the class must implement *all* required methods
- The protocol determines what is required versus optional
- Protocols can inherit from other protocols

Creating a Protocol Example

```
protocol Deliverable {
  var deliveryTime: Date {get set}

  func dropOffInstructions()

}
```

Implementing a Protocol Example

```
class Pizza: Food, Deliverable {

   var deliverTime: Date

   init() {

      deliveryTime = Date()

   }

   func dropOffInstructions() {

      print("Hand pizza box directly to customer")

   }

}
```

Why Use Protocols?

- More flexible than normal class interface
  - Reuse single API declaration in unrelated classes
- Clear design and purpose in code structure
  - Specify an object's role across application
  - Ensures consistency across objects that share this functionality

Reuse of Protocols Example

```
class Pizza: Food, Deliverable {

   var deliveryTime: Date

   init() {

      deliveryTime = Date()

   }

   func dropOffInstructions() {

      print("Hand pizza box
      directly to customer")

   }

}
```

```
class Book: Deliverable {

   var deliveryTime: Date

   init() {

      deliveryTime = Date()

   }

   func dropOffInstructions() {

      print("Leave box within mailbox
      if it fits, or next to the
      mailbox if it does not”)

   }

}
```

UITableViewDataSource Redux

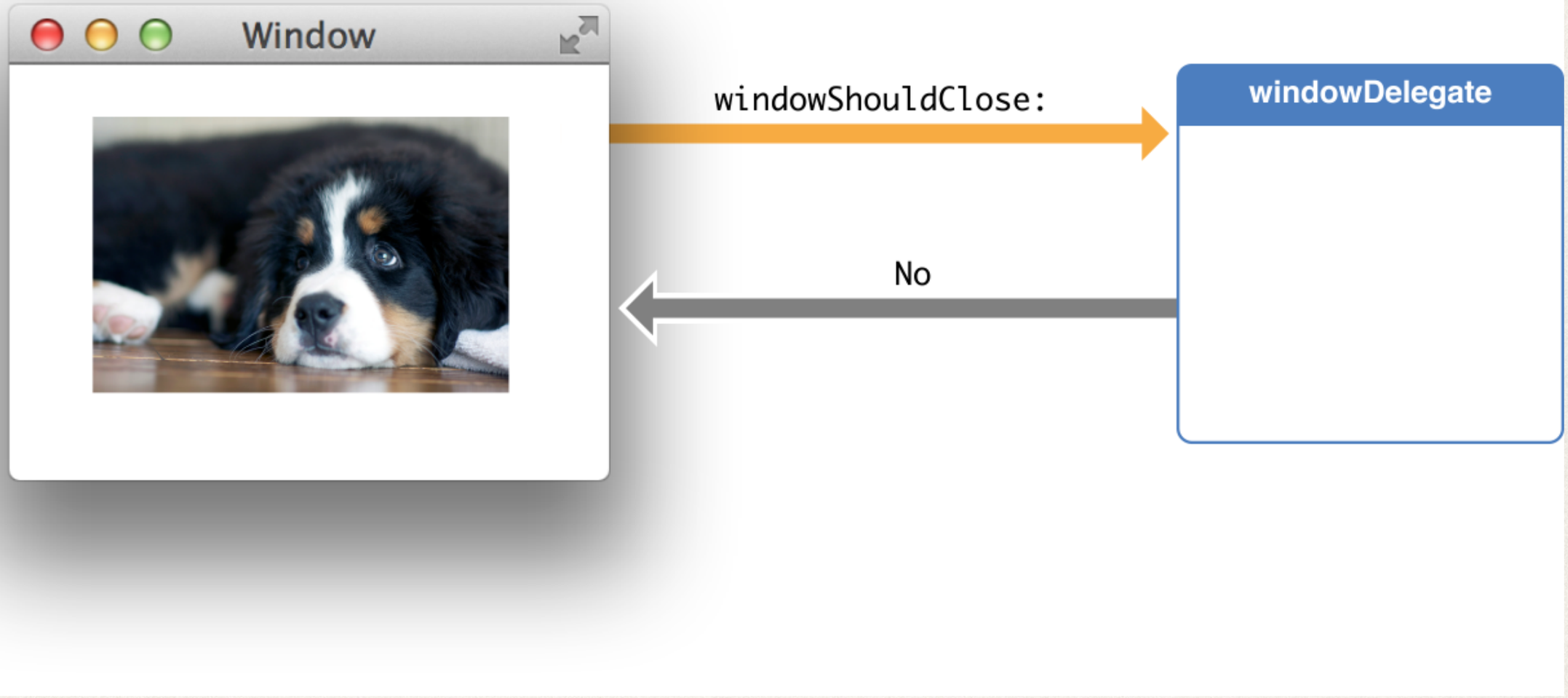
```
protocol UITableViewDataSource : NSObjectProtocol {
. . . . .

    func tableView(UITableView, numberOfRowsInSection: Int) -> Int
     /* Data source returns the number of rows in a given section of table
  view */

    func tableView(UITableView, cellForRowAt: IndexPath) -> UITableViewCell
    /* Data source returns cell to insert in particular location of table
    view */

    optional func tableView(UITableView, canEditRowAt: IndexPath) -> Bool
    /* Optional: Data source determines whether row is editable */

    optional func tableView(UITableView, canMoveRowAt: IndexPath) -> Bool
    /* Optional: Data source determines whether row is movable */


. . . . .

}
```

Delegates

- Pattern where one object acts *on behalf of* (or in coordination with) another object
- Allows for customization of several objects' behavior via one central object
- Simplifies communication between objects
- Used extensively in iOS development
- Closely associated with protocols

How Delegates Work

- Delegating object keeps *reference* to delegate
- Sends message to delegate at appropriate time
- Delegate returns with message at appropriate time
- Multiple delegates allowed per delegating object


Window
windowShouldClose:
windowDelegate
No

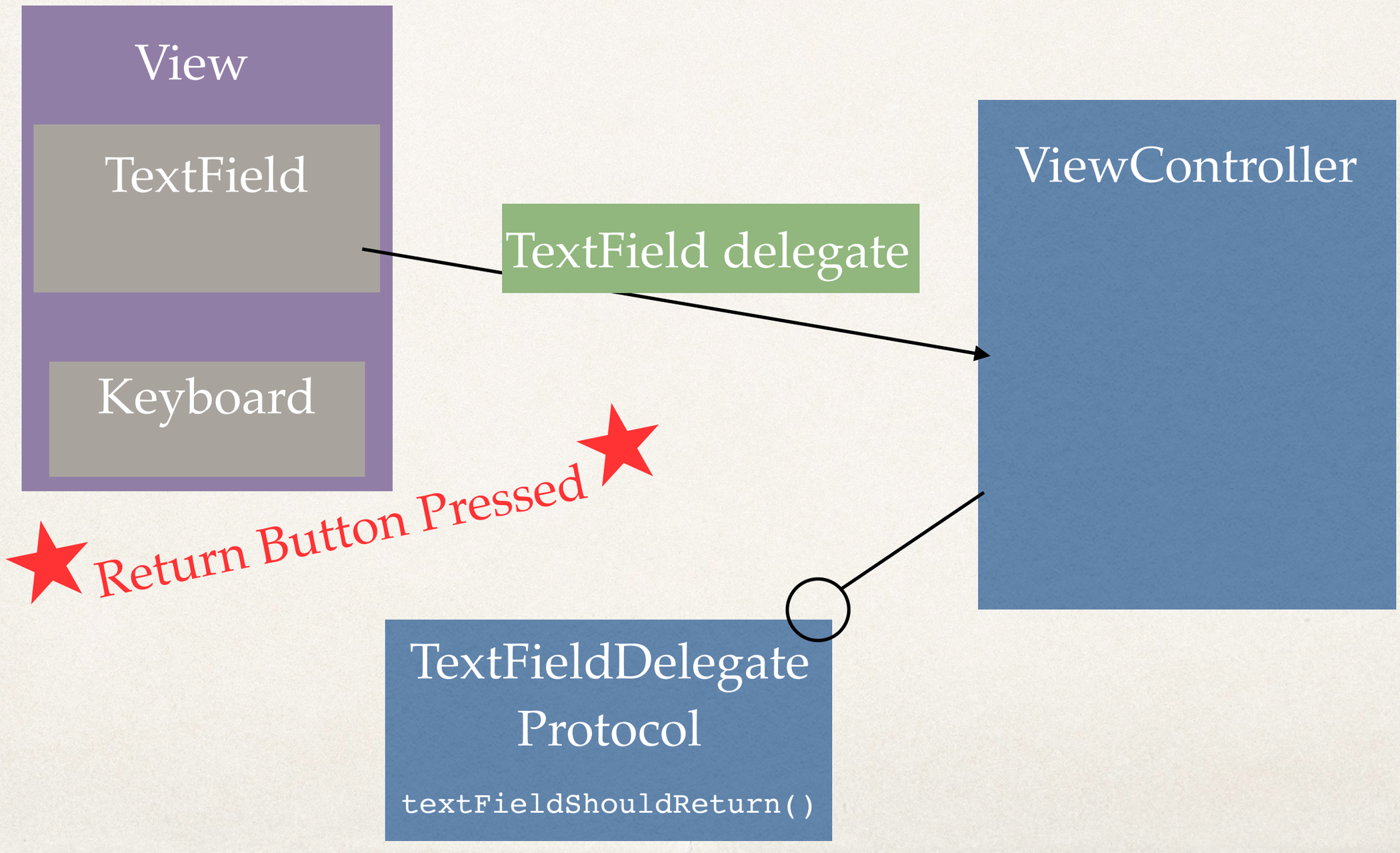
UITextFieldDelegate Example

```
class ViewController: UIViewController, UITextFieldDelegate {

    override func viewDidLoad() {

        super.viewDidLoad()

        myTextField.delegate = self

    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {

        textField.resignFirstResponder()

        return true

    }

}
```

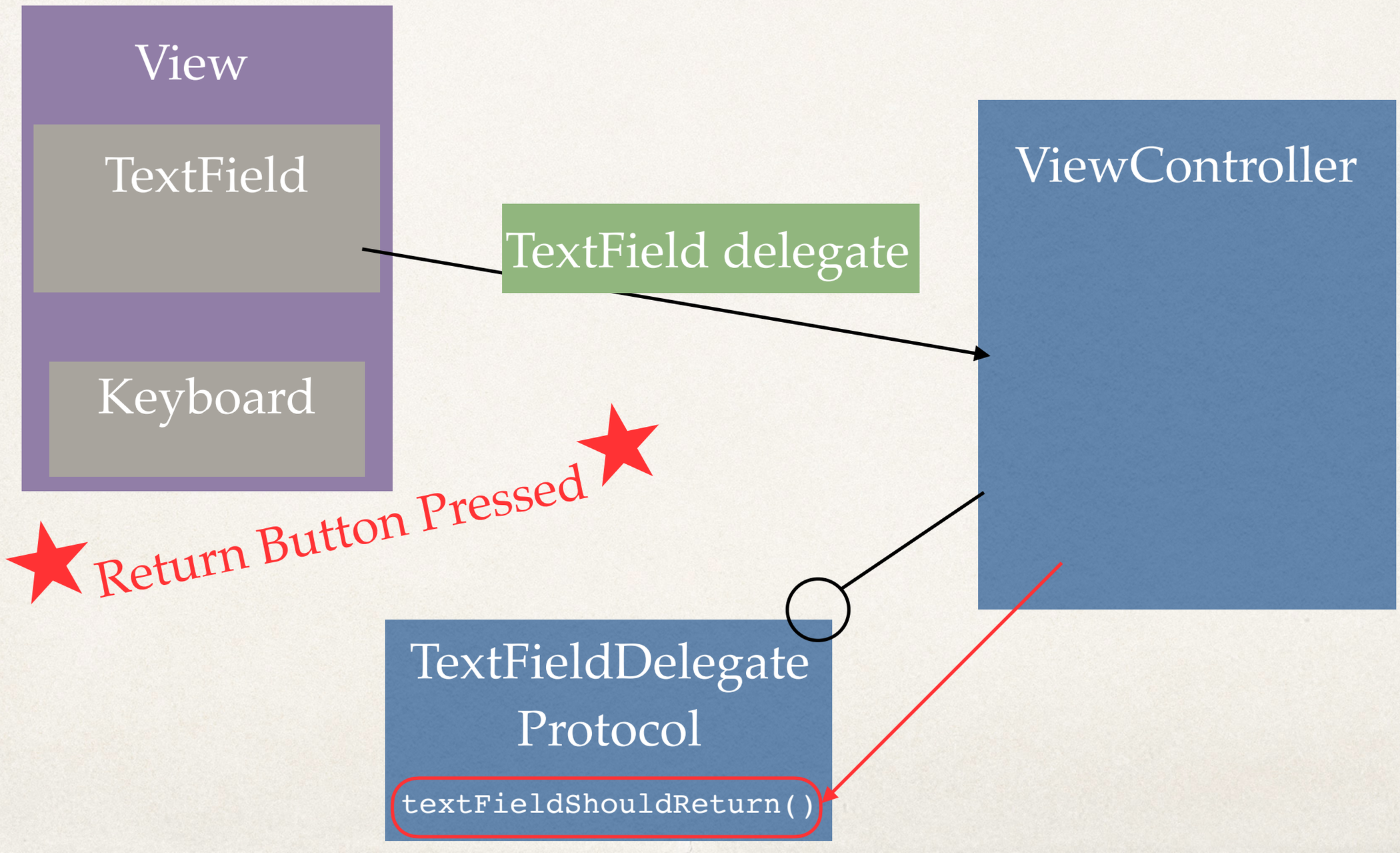
UITextFieldDelegate Protocol

- UITextFieldDelegate is a protocol
  - Implemented by the View Controller
- Manages the editing and validation of text in any Text Field object
  - All protocol functions are optional
- `textFieldShouldReturn()` is one of these protocol functions
  - Asks the text field's *delegate* if the text field should process the pressing of the "Return" button
  - Text field has delegate call this function when "Return" button is pressed
    - Allows a ViewController to determine text handling rather than modifying the text field implementation directly

Putting It Together

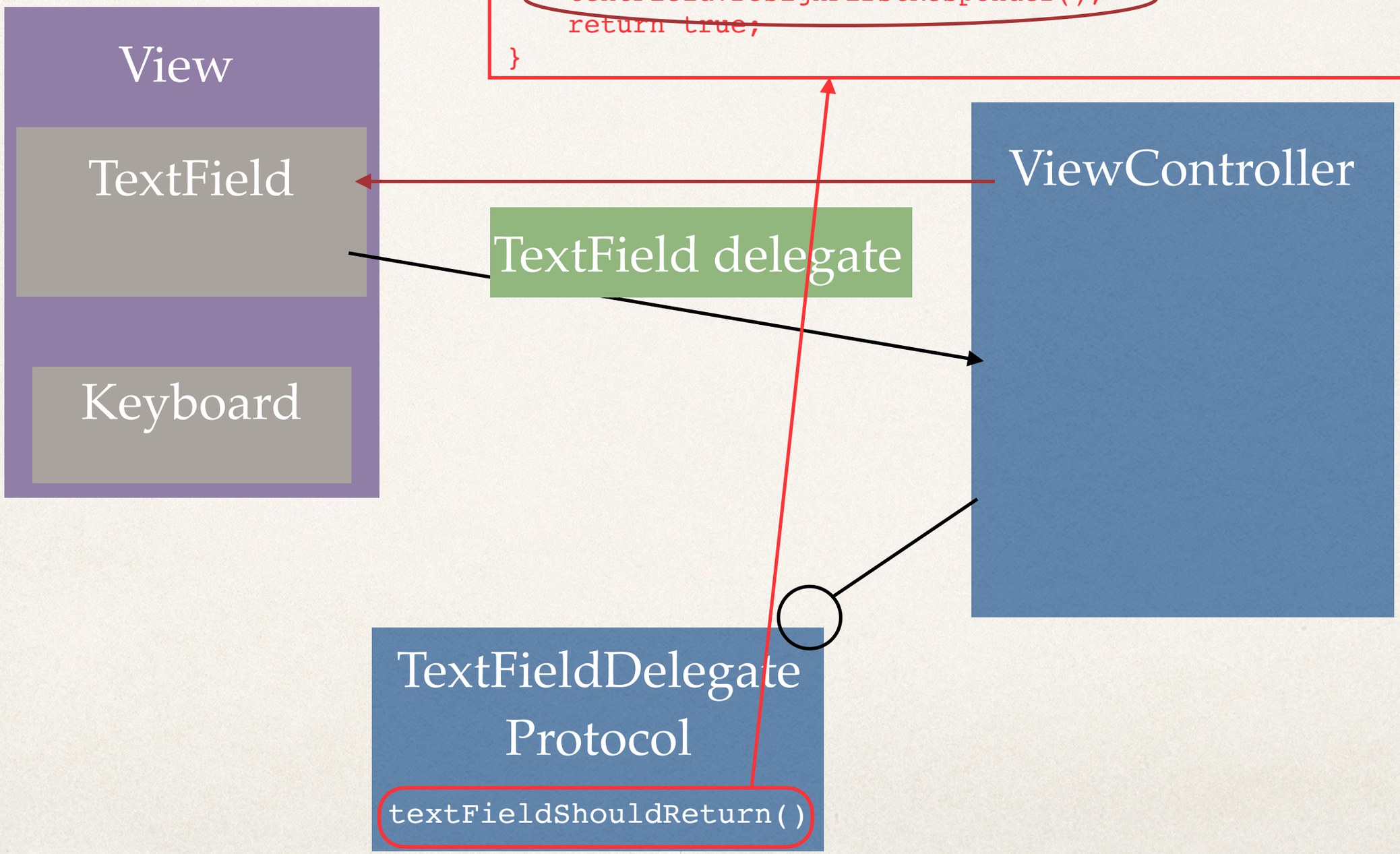


Putting It Together



Putting It Together

```
func textFieldShouldReturn(_ textField: UITextField) -> Bool {
    textField.resignFirstResponder();
    return true;
}
```



Delegates in iOS

- Used everywhere!
- Allow UIViews to communicate with UIViewControllers
- Allow asynchronous handling of network data
- Manages lifecycle of app (UIApplicationDelegate)
  - Handles opening and closing of app
  - Performs memory management within app

Quiz Question: Protocols and Delegates

- What is the difference between a protocol and a delegate?

  - A protocol acts on behalf of another object; a delegate is functionality implemented by any class

  - A protocol is functionality implemented by the delegate; a delegate communicates between objects

  - A protocol is functionality implemented by any class; a delegate acts on behalf of another object

  - A protocol communicates between objects; a delegate is functionality implemented by the protocol