Protocols and Delegates

Dr. Sarah Abraham
Maintaining a Specification

- Remaining abstract means conforming classes must meet some specification

- C++ uses abstract classes
  - Methods not implemented in this base class
  - Subclasses must implement all “virtual” methods that have same input/output as abstract class

- Java uses interfaces
  - Same concept as abstract classes in C++
  - More rigid implementation to ensure conformity

- Swift uses protocols
  - Same concept as abstract classes and interfaces
  - Allows for greater flexibility for extending a protocol to have additional functionality or requirements
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

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

Class specifications 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 protocol in order to implement it
- Classes that conform to protocol must implement all required methods
- Setup of protocol determines what is required versus optional
- Protocols can inherit from other protocols
Creating a Protocol Example

```swift
protocol Deliverable {
    var time: Date {get set}
    func dropOffInstructions()
}
```
class Pizza: Food, Deliverable {

    var time: Date

    init() {
        time = Date()
    }

    func dropOffInstructions() {
        print("Hand 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
Reuse of Protocols Example

class Pizza: Food, Deliverable {
    var time: Date
    init() {
        time = Date()
    }
    func dropOffInstructions() {
        print("Hand directly to customer")
    }
}
class Book: Deliverable {
    var time: Date
    init() {
        time = Date()
    }
    func dropOffInstructions() {
        print("Leave by mailbox")
    }
}
protocol UITableViewDataSource : NSObjectProtocol {

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

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

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

    optional func tableView(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
windowShouldClose:
No
windowDelegate
UITextFieldDelegate Example

```swift
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
Putting It Together

View

TextField

TextField delegate

ViewController

UITextFieldDelegate Protocol

textFieldShouldReturn()
Putting It Together

View

TextField

Keyboard

ViewController

TextEdit delegate

TextFieldDelegate Protocol

textFieldShouldReturn()
Putting It Together

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

- Classes can use both protocols and delegates

![Diagram]

- `UITableViewController` is subclassed to your class.
- Calls come from the base `UITableViewController` class down to your class.
- Conforms to: `UITableViewDelegate`, `UITableViewDataSource`.
- Delegate of `UITableViewController`.

Your table view controller class
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
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