Protocols and Delegates

Ana Nogal
@anainogal
@iOSStepByStep
http://ananogal.com

Agenda

What are we learning today?

- ***Protocols***
- ***Delegates***
- ***Model View Controller***
- ***Scroll Views***

What is a Protocol?

A protocol defines a blueprint of methods, properties, and other requirements that suit a particular task or piece of functionality. The protocol can then be adopted by a class, structure, or enumeration to provide an actual implementation of those requirements. Any type that satisfies the requirements of a protocol is said to conform to that protocol.

– **The Swift Programming Language Guide by Apple**

What?!?!!?!?!

Protocol

Defines a set of required functionality that other types can adopt.

Declaring a Protocol

```
protocol NameOfTheProtocol {

    //body

}
```

Naming a Protocol

- **Protocols that describe *what something is* should read as nouns** (e.g. `Collection`).
- **Protocols that describe a *capability* should be named using the suffixes** `able`**,**`ible`**, or** `ing` (e.g. `Equatable, ProgressReporting`).

Swift Design Guidelines

Using a Protocol

```
protocol Flyable {
    func fly()
}

struct Plane: Flyable {
    var brand: String
}   func fly() {
        print("I can fly")
    }
}
```

Challenge

Create a protocol named “**Flyable**” with a method fly

Create a class named **Vehicle** that has a property ***numberOfWheels*** and a method ***startMoving***

Create a **Plane** class that inherits from **Vehicle** and conforms to the **Flyable** protocol

Create a class **Bird** that conforms to the **Flyable** protocol.

What have we learned so far?

- ***Protocols***

  They define a set of required functionality that other types can adopt.

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.

Apple Documentation

What?!?!!?!?!

Implementing a Delegate

- Create a delegate protocol that defines the responsibilities of the delegate.
- Create a delegate property in the delegating class to keep track of the delegate.
- Adopt and implement the delegate methods in the delegate class.
- Call the delegate from the delegating object.

Demo

Delegates

Delegates: Step 1

Create a delegate protocol that defines the responsibilities of the delegate.

```
protocol TitleDelegate: class {
    func didChangeTitle(title: String)
}
```

Delegates: Step 2

Create a delegate property in the delegating class to keep track of the delegate.

```
class TitleOwner {
  var title: String = “”
  weak var delegate: TitleDelegate?
}
```

Delegates: Step 3

Adopt and implement the delegate methods in the delegate class.

```swift
class TitleBillboard: TitleDelegate {
    func didChangeTitle(title: String) {
        print("The title in the TitleOwner class is \(title)")
    }
}
```

Delegates: Step 4

Call the delegate from the delegating object.

```
class TitleOwner {
    var title: String = ""
    weak var delegate: TitleDelegate?

    func setNewTitle(title: String) {
        self.title = title
        delegate?.didChangeTitle(title: title)
    }
}
```

Use it !

In the Playground do:

```
var titleBillboard = TitleBillboard()
var titleOwner = TitleOwner()
titleOwner.delegate = titleBillboard

titleOwner.setNewTitle(title: "My new title ")
```

Power of Protocols

Create another class that conforms to the TitleDelegate protocol.

```
class AnotherTitleClass: TitleDelegate {
    func didChangeTitle(title: String) {
        print(“\(title) is the title in TitleOwner”)
    }
}
```

Power of Protocols

In the Playground do:

```
var titleBillboard = TitleBillboard()
var titleOwner = TitleOwner()
titleOwner.delegate = titleBillboard

titleOwner.setNewTitle(title: "My new title “)

var anotherTitleClass = AnotherTitleClass()
titleOwner.delegate = anotherTitleClass
```

App Life Cycle

A good example of a delegate is the AppDelegate class.

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
   return true
}

func applicationWillResignActive(_ application: UIApplication) {}

func applicationDidEnterBackground(_ application: UIApplication) {}

func applicationWillEnterForeground(_ application: UIApplication) {}

func applicationDidBecomeActive(_ application: UIApplication) {}

func applicationWillTerminate(_ application: UIApplication) {}
```

Challenge

Create project and files:

Create a new project, call it “***DelegatedInAction***” and delete the ***ViewController.swift*** file.

Create 3 new files:

- Create a Swift file called “***TitleDelegate***”
- Create a CocoaTouch, Subclass UIViewController and call it “***TitleBillboardViewController***”
- Create a Cocoa Touch, Subclass UIViewController and call it “***TitleOwnerViewController***”

Challenge

Delegate:

- Copy the protocol from your playground and paste it in the ***TitleDelegate.swift*** file
- In the ***TitleOwnerViewController***, add the delegate variable (You don't need to add the title variable, since the ViewController already has one)
- Go to ***TitleBillboardViewController.*** Make it conform to the ***TitleDelegate*** Protocol.

Challenge

Main.Storyboard:

- Select the ***ViewController***, go to the Identity inspector (3rd button from the left), and set the class property to ***TitleBillboardViewController***
- Add a new ***ViewController*** to the storyboard, and set it’s class to ***TitleOwnerViewController***

Challenge

Main.Storyboard:

- Go to ***TitleBillboardViewController*** and add:
  - a label, center it, set it’s text to “Placeholder for title”
  - add a IBOutlet for the label.
  - a button, center it, set it’s text to “Set title”

- Go to ***TitleOwnerViewController*** and add:
  - a textField, center it and set it’s placeholder property to ”Set the new title to this controller”
  - set the background of the controller to “Light Grey”
  - add a IBOutlet for the textField
  - add a IBAction to the textField
  - add a button with title: “Back”

Challenge

TitleBillboardViewController.swift:

Create a method to unwind the segue:

```swift
@IBAction func dismiss(segue: UIStoryboardSegue)
```

Challenge

Main.Storyboard:

- Make a segue “Present modally” from the “Set Title” button on ***TitleBillboardViewController*** to the ***TitleOwnerViewController***
- In the Attributes Inspector set the segue identifier to “GoToTitleOwner”
- In the ***TitleOwnerViewController*** make the button Trigger the unwind segue

Challenge

TitleBillboardViewController.swift:

In the ***TitleBillboardViewController*** uncomment the function:
override func prepare(for segue: UIStoryboardSegue, sender: Any?)

Set a var “destinationController” with the destination controller from the segue.
Set the delegate of the destinationController to self.

Challenge

TitleOwnerViewController.swift:

In the IBAction, get the text from the textField and call the delegate and pass it as the title parameter.

TitleBillboardViewController.swift:

Find the method from the protocol and in the body set the text of the label to “The new title is: \(title)”

Challenge

Run and build the app
Set a new title in the text field
Come back and see the label text updated.

What have we learned so far?

- ***Protocols***

  They define a set of required functionality that other types can adopt.

- **Delegates**

  Delegates allow one object to send messages to another object when a specific event happens.