Protocols & Delegates

Advanced Topics in iOS & Swift
11/14/16

Adding a Friend

1. add *property* of type `FriendsTableViewController` in the `AddFriendViewController` so that you can *send a message* to it

2. create a new *method* in `FriendsTableViewController` that the `Friend` that is passed in gets added to the `friends` array and the `tableView` gets updated afterwards

3. in `prepare(for segue:)` in `FriendsTableViewController`, assign the *property* you created in 1. on the `AddFriendViewController` that is the `destination` of the `segue`

4. call the *method* you created in 2. on the *property* you created in 1. and pass a `Friend` that you created based on the user input in the *text field* and the *segmented control*

Protocols

allows to ***ensure*** **that a class implements a certain function** (*contract*)

can have ***required*** **and** ***optional*** **methods**

essential tool for implementing the ***delegate pattern***

Protocol Example

```
protocol Computable {
    func compute() -> Int
}
```

```
class Sum {
  var numbers = [1,2,3,4,5]
}
```

Protocol Example

```
protocol Computable {
    func compute() -> Int
}
```

```
class Sum {
  var numbers = [1,2,3,4,5]
}
```

```
class Sum: Computable {
  var numbers = [1,2,3,4,5,6,7,8,9]
}
```

Type 'Sum' does not conform to protocol 'Computable'

Protocol Example

```
protocol Computable {
    func compute() -> Int
}
```

```
class Sum {
  var numbers = [1,2,3,4,5]
}
```

```
class Sum: Computable {
  var numbers = [1,2,3,4,5,6,7,8,9]

  func compute() -> Int {
    var sum = 0
    for number in numbers {
      sum = sum + number
    }
    return sum
  }
}
```

Delegates

a delegate is always ***defined*** ***by a protocol***

decouples your code (makes it so that the ***view*** **doesn't know anything about its *controller***)

***view* uses delegate** to call functions without knowing what the actual class is (only needs to know that it *conforms* to protocol)

Two Use Cases

1. **Using A D*elegate*** - you want to get access to another class by *becoming* its *delegate* (e.g. all *delegates* from `UIKit` or other Apple frameworks)

2. **Creating Your Own** - you want to provide access to your class without knowing anything about the class that will use it (e.g. our `FriendsTableViewCell` example)

[3. **Both**]

Using a Delegate

<u>3 Simple Steps</u>

1. make class conform to the *delegate* protocol by adding it to the class declaration

(e.g. `ViewController: UIViewController, TableViewDelegate`)

2. implement the required methods that are specified in the delegate protocol

3. *assign* your class to be the *delegate* of the object that needs the delegate

(e.g. `tableView.delegate = self`)

Creating Your Own

3 Simple Steps

1. write the protocol and specify required and optional methods

2. create the `delegate` property inside the class that needs the delegate

3. call the (required and optional) methods on the `delegate`