protocols / delegates / getting values from UI

CS112 Unit 9

Max Luttrell, Fall 2016

protocols

- a **protocol** specifies one or more requirements that a type must fulfill
- a type which satisfies these requirements is said to **conform** to the protocol

protocols - syntax

```
// define a protocol
protocol SomeProtocol {
    // protocol definition goes here
}

// define another protocol
protocol SomeOtherProtocol {
    // protocol definition goes here
}

// a class which conforms to SomeProtocol
class SomeClass: SomeProtocol {
    // class definition goes here
}

// a class which has a superclass and also conforms to
// SomeProtocol
class SomeOtherClass: SomeSuperClass, SomeProtocol {
    // class definition goes here
}
```

protocol requirements - property

- a protocol can specify property requirements and/or method requirements

- a property requirement can be specified as **gettable**, or **gettable/settable**

```
protocol hasName {
    var name: String {get set}
}
```

protocol example - property

- class Player conforms to protocol hasName, because it has property var name

```
protocol hasName {
    var name: String {get set}
}

class Player: hasName {
    var name = ""
    var weight = 0.0
    var height = 0.0
    var age = 0

    func printInfo() {
        print("\(name)")
        print("\(weight) kg, \(height) m, \(age) yrs")
    }
}
```

protocol requirements - method

- a protocol can also specify method requirements

```
protocol printsInfo {
    func printInfo()
}
```

protocol example - method

- class Player also conforms to protocol printsInfo, because it has method printInfo()

```
protocol hasName {
    var name: String {get}
}

protocol printsInfo {
    func printInfo()
}

class Player: hasName, printsInfo {
    let name = ""
    var weight = 0.0
    var height = 0.0
    var age = 0

    func printInfo() {
        print("\(name)")
        print("\(weight) kg, \(height) m, \(age) yrs")
    }
}
```

Exercise 9A

- modify the below Player class so that it conforms to the builtin **`CustomStringConvertible`** protocol. a class conforming to this protocol must have a gettable property named **`description`**

- create a player object, and then call print on your object to print it out

```
class Player {
    var name: String
    var weight: Int
    var height: Int
    var age: Int

    func printInfo() {
        print("\(name)")
        print("\(weight) kg, \(height) m, \(age) yrs")
    }
}
```

delegation

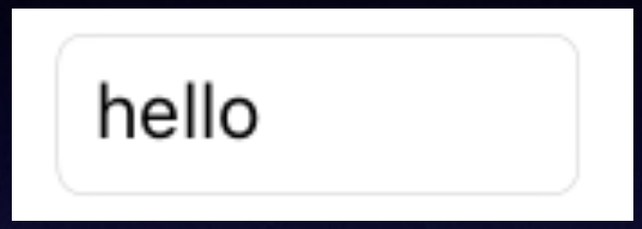
- an object can hand off some of its tasks to another object - this is known as **delegation**

delegation

- in Swift, we can guarantee that a delegate can handle its delegated responsibilities by making it conform to a protocol
- delegation is used often in iOS

delegation example - UITextField

- consider a text field in an iOS app:



- this is implemented with a **`UITextField`** object

- to enter text, the user taps the text field. the **`UITextField`** object becomes the **first responder** of the device, i.e. the user input goes to it

- when the user is done entering some text, the **`UITextField`** object can let a delegate know, and the delegate can decide if it wants to resign as first responder

delegation example - UITextField

- we can set up our ViewController class to be the delegate of the **`UITextField`** object! this involves a couple things:
  - it must conform to the **`UITextFieldDelegate`** protocol.
  - all of the methods in this protocol are **optional**. we will implement these methods:

```
func textFieldShouldReturn(textField:
UITextField) -> Bool

func textFieldDidEndEditing(textField:
UITextField)
```

- it must set itself to be the delegate of the **`UITextField`** object

demo - build an app with a UITextField

Exercise 9B

- we have built an app which has a label and a text field. when the user hits return, the label changes to the text the user entered

- append an exclamation point to the text that the user entered. for example, if the user enters “Giants” the label should change to “Giants!” after the user is done entering.

- add another label to the Storyboard. after the user enters some text, change this label to the number of characters the user entered. see the hint below how to get the numbers of characters in a String

```
// if we have the following string
let str = "giants"
// the below will give us 6
str.characters.count
```