Protocols, Delegates, and Segues

Protocols

- A *protocol* is the declaration of a group of related properties, initializers, and methods that provide a desired task or level of functionality.
- Some other languages refer to them as *interfaces*.
- When a class implements the properties, initializers, and methods of a protocol, it is said to *adopt* or *conform* to the protocol.
- The parts of a protocol can be implemented in any class. As such, they are independent of any class.
- When you define a protocol, you identify which parts are required and which parts are optional. When a class conforms to a protocol, all *required* methods must be implemented.
- Just like classes, protocols can inherit from other protocols.

Syntax

The syntax of a protocol looks very much like that of a class or struct:

```
protocol <ProtocolName> {
    // definition of protocol
}
```

Syntax

For a type to adopt the protocol, you list the protocol name after the colon in the type definition:

```
class ClassName: Protocol1, Protocol2 {
    // class definition
}
```

Note that a type can adopt multiple protocols.

Syntax

If a class has a superclass, it must go first:

```
class ClassName: MySuperClass, Protocol1,
    Protocol2 {

    // class definition
}
```

Example

Here's the definition of a protocol called `Resizable`:

```
protocol Resizable {

    var width: Float { get set }
    var height: Float { get set }

    init(width: Float, height: Float)

    func resizeBy(wFactor: Float,
                  hFactor: Float)
}
```

When a protocol requires a property, it provides the name and type, and indicates whether the property is gettable, settable, or both.

Example (cont.)

Here's a class called Rectangle that conforms to the `Resizable` protocol:

```
class Rectangle: Resizable {
    var width: Float
    var height: Float

    required init(width: Float, height: Float) {
        self.width = width
        self.height = height
    }
    func resizeBy(wFactor: Float, hFactor: Float) {
        width *= wFactor
        height *= hFactor
    }
}
```

Example (cont.)

```
class Rectangle: Resizable {
    var width: Float
    var height: Float
    var description: String {
        return "Rectangle, width \(width), height
            \(height)"
    }
    required init(width: Float, height: Float) {
        self.width = width
        self.height = height
    }
    func resizeBy(wFactor: Float, hFactor: Float) {
        width *= wFactor
        height *= hFactor
    }
}

let rect = Rectangle(width:10,height:20)
rect.resizeBy(wFactor:2,hFactor:2)
```

Example (cont.)

```
class Rectangle: Resizable, CustomStringConvertible {
    var width: Float
    var height: Float
    var description: String {
        return "Rectangle, width \(width), height
            \(height)"
    }
    required init(width: Float, height: Float) {
        self.width = width
        self.height = height
    }
    func resizeBy(wFactor: Float, hFactor: Float) {
        width *= wFactor
        height *= hFactor
    }
}

let rect = Rectangle(width:10,height:20)
rect.resizeBy(wFactor:2,hFactor:2)
print(rect)   // prints "Rectangle, width 20.0, height 40.0"
```

Multiple inheritance

Note that protocols are a way to have *fake* multiple inheritance.

It is a way to define a set of additional methods a class must (required) or could (optional) implement, since neither Swift nor Objective-C support multiple inheritance.

Delegates

A *delegate* is a simple but powerful pattern in which one object acts on behalf of or in coordination with another object.

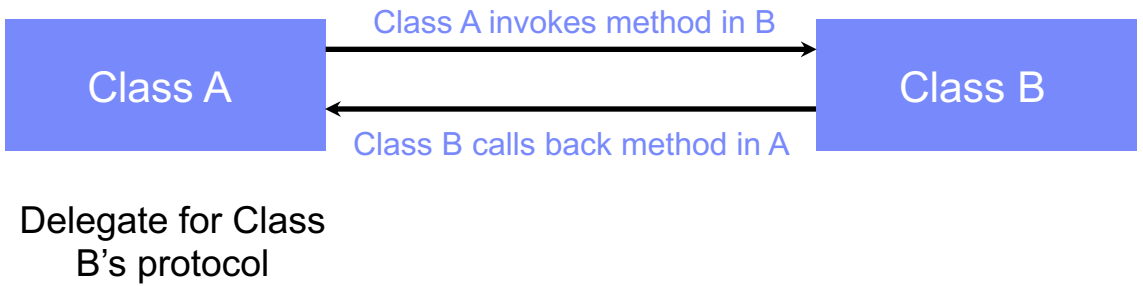
- The delegating object keeps a reference to another object (the delegate) and at the appropriate time, sends a message to it. The main value of delegation is that it allows you to easily customize the behavior of several objects in one central object.
- There's nothing that says you can't have more than one delegate.

Delegates (cont.)

A delegate is a pointer to *some object* that has implemented the protocol's methods.

- The "some object" means we don't really know *or care* specifically what kind of object the delegate is referring to – only that the methods defined in the protocol are implemented in that object.

Class A conforms to a protocol needed by Class B
Class B calls a method of the protocol that lives in Class A's object



Delegates (cont.)

Protocol

- Will do [x]
- Will do [y]
- Will do [z]

Delegate

Conforms to protocol…

- Can dependably do [x] if asked
- Can dependably do [y] if asked
- Can dependably do [z] if asked

Delegator

Needs a delegate who can dependably

- Do [x] when asked
- Do [y] when asked
- Do [z] when asked

Segue

A *segue* is a named transition between one part of the UI to another. Its purpose is to make it easier to move from one VC to another.

- A segue is created in IB and code is written to make use of it.