ATLS 4120/5120: Mobile Application Development
Week 5: Delegation

What do you think of when you hear the word delegation?
1. The verb, "to delegate", meaning "to give control"
2. The noun, "a delegate", meaning "a person acting for another"
3. The made-up noun, "a delega*tor*", or more properly, a *principal*, meaning "a person who delegates to another"

A delegator/principal (noun) would delegate (verb) control or responsibility to another person called a delegate.

When planning a party you can plan and buy everything yourself or you can delegate some of the tasks to someone else.

Delegates

In iOS delegation is a pattern where one class has given another class responsibility for some tasks. That class is its delegate.

- Delegation is a common design pattern used in Cocoa Touch.
- Many UIKit classes allow customization of their behavior through delegation.
- Delegation enables objects to take responsibility for doing certain tasks on behalf of another object.

Protocols
- For a class to act as a delegate it needs to conform to a protocol.
- A protocol is similar to a class in that it can define both methods and properties for certain functionality
    - A protocol can contain both required and optional methods
    - They are a template because they don't provide the implementation, that's up to the class that adopts the protocol
- To the delegator class, the protocol is a guarantee that some behavior will be supplied by the delegate.
- The protocol is a set of obligations – things that must be implemented when the protocol is adopted.
- When a class adopts a protocol it must implement its required methods so its objects can respond to those methods.

So there are three pieces involved in the delegate/protocol pattern
1. A protocol defining the responsibilities that will be delegated
2. A delegator, which depends on an instance of something conforming to that protocol
3. A delegate, which adopts the protocol and implements its requirements

There are 3 steps in implementing a protocol in a class: (slide)
1. Adopt the delegate protocol
2. Implement the delegate methods.
3. Set the controller as the delegate

Two classes that use the delegation pattern are
- UITextField class
    - A text field is a single line area for entering text.
    - UITextFieldDelegate protocol
    - The delegate for UITextField is notified when a key is tapped and the keyboard automatically slides up from the bottom of the screen or the keyboard is dismissed.

- UIAlertController class
    - Alerts are primarily used to inform the user of something important or verify a destructive action.
    - alerts appear in a small, rounded view in the center of the screen on both iPhone and iPad
    - Alerts interrupt the user experience so use them sparingly
    - preferredStyle: UIAlertControllerStyle.Alert
    - Action sheets display a list of 2 or more choices to the user when a toolbar button is tapped.
    - Users are unable to continue until they chose one of the choices
    - preferredStyle: UIAlertControllerStyle.ActionSheet
    - On the iPhone, the action sheet always pops up from the bottom of the screen.
    - On the iPad, it's displayed in a **popover**—a small, rounded rectangle with an arrow that points toward another view, usually the one that caused it to appear.
    - To present an alert or action sheet
        1. Create a UIAlertController object with the title, message and preferredStyle you want
        2. Define your UIAlertAction objects
            - 1 for each button in the alert/action sheet
        3. Add your UIAlertAction objects to your UIAlertController object
        4. Call presentViewController() to present your alert or action sheet

Tip Calculator
(tipcalculator)
Create a single view application, iPhone (or iPad), named tipcalculator.

Go into the storyboard and add 3 text fields with labels next to them(check amount, tip %, # of people), and then 3 labels(tip, total, total per person) with labels next to those to match mine. Think about where you should put the text fields so users can use the keyboard.
Text fields should have a font size of around 15 and use right alignment. Set the keyboard for the textfields to numbers and punctuation (look at number pad as well).
Set some defaults if you want.
You can also set the text fields to clear when you start editing (check box).
You can also customize the return key to say Done if you want.

Now let's make the connections for the text fields and labels on the right. They are all outlets.
checkAmount, tipPercent, people, tipDue, totalDue, totalDuePerPerson
Now these variables are defined in our swift file.

If you run it you might not see some of your fields.
Go back into the storyboard to set the constraints
Try Add Missing Constraints

If you don't see your bottom row of labels remove the constraint they each have to the bottom layout guide. Now they should appear.
If you move any elements you need to update their constraints
This at least helps in portrait mode for now so we can keep working. We can tweak the constraints as needed later.

Try taping one of the text fields. Even with no code the keyboard shows up but you can't get rid of it since this keyboard doesn't have a dismiss keyboard button.
We need to adopt the UITextField protocol.
```
class ViewController: UIViewController, UITextFieldDelegate
```

Implement the UITextFieldDelegate method that is called when the return button is pressed
```
    func textFieldShouldReturn(textField: UITextField) -> Bool {
        textField.resignFirstResponder()
        return true
    }
```

Assign our controller as the delegate for each UITextField. Then the text field will tell its delegate when certain events, like the return key being tapped, take place.

```
override func viewDidLoad() {
        checkAmount.delegate=self
        tipPercent.delegate=self
        people.delegate=self
        super.viewDidLoad()
}
```
viewDidLoad() is called automatically after the views have been initialized but before it's displayed.
Run the app and see if the return key dismisses the keyboard.
If you also want to dismiss the keyboard by tapping the background, the book covers it in chapter 4.

Note where your keyboard comes up. Is it blocking any of the text fields? Do you need to fix the layout of your UI so it works with a keyboard?

(remove text from the computed labels)

Now it's time to write the method that calculates the tip, total price, and price per person.
```
    func updateTipTotals() {
        var amount:Float //check amount
        var pct:Float //tip percentage

        if checkAmount.text!.isEmpty {
           amount = 0.0
        } else {
           amount = Float(checkAmount.text!)!
        }
        if tipPercent.text!.isEmpty {
           pct = 0.0
        }
        else {
           pct = Float(tipPercent.text!)!/100
        }
```

```swift
        let numberOfPeople=Int(people.text!) //returns an optional
      let tip=amount*pct
       let total=amount+tip
       var personTotal : Float = 0.0 //specify Float so it's not a Double
       if numberOfPeople != nil {
           if numberOfPeople! > 0 {
               personTotal = total / Float(numberOfPeople!)
           }
       }
       //format results as currency
        var currencyFormatter = NSNumberFormatter()
      //set the number style
        currencyFormatter.numberStyle=NSNumberFormatterStyle.CurrencyStyle
      //returns a formatted string
        tipDue.text=currencyFormatter.stringFromNumber(tip)
        totalDue.text=currencyFormatter.stringFromNumber(total)
      totalDuePerPerson.text=currencyFormatter.stringFromNumber(personTotal)
    }
```

When should we call this method?
We want the tip and totals fields to be updated when any of the text fields are changed. We could use textFieldShouldReturn but that is only called when the user hits Return. What if they just click in one text field and then another?
The UITextFieldDelegate has another method called textFieldDidEndEditing that is called when the user finishes editing a text field (pressing Done or switching to another field). So let's call updateTipTotals from textFieldDidEndEditing so the tip labels will update as soon as the user finishes editing any text field.

```swift
    func textFieldDidEndEditing(textField: UITextField) {
        updateTipTotals()
    }
```

Now that your labels have values if they're not lined up, select them all and add a leading edge alignment.

Alert
Although we want to use alerts sparingly because they interrupt the user experience, using them to warn the user of errors is a good use of them.
So let's add an alert if the user has number of people at 0.

We update updateTipTotals (add else statement to the if)
```swift
else {
    //create a UIAlertController object
    let alert=UIAlertController(title: "Warning", message: "The number of
people must be greater than 0", preferredStyle:
UIAlertControllerStyle.Alert)
    //create a UIAlertAction object for the button
    let cancelAction =UIAlertAction(title: "Cancel",
style:UIAlertActionStyle.Cancel, handler: nil)
    //add the alert action to the alert object
    alert.addAction(cancelAction)
```

```
        let okAction =UIAlertAction(title: "OK", style:
UIAlertActionStyle.Default, handler: { action in
                        self.people.text="1"
                        self.updateTipTotals()
                    })
    alert.addAction(okAction)
    //present alert
    presentViewController(alert, animated: true, completion: nil)
            } //end else
```

The handler parameter takes a closure. A closure is a block of code, like a function without a name.
(You can actually write a function instead and just call it here) This is similar to an anonymous function.
This is called when the user selects the action.

Test it with different values for people.
(You might get a warning, just ignore it.)

Update your launch screen and add app icons

Try the app in landscape for different size phones, does it work?
If you can't get the constraints to work for all, change the size class to Compact hCompact and modify
the constraints for iPhone landscape.

Lab: Create an app that includes multiple textfields, does a calcluation, and shows an alert or action
sheet for some condition. The keyboard should be dismissed when the user taps the Done/Return key or
touches the background view(hint: chapter 4 of the book) from all textfields.. Your app should work for
all size iPhones in portrait and landscape orientation. Don't forget your app icons and launch screen.