**ATLS 4120/5120: Mobile Application Development**
**Week 5: Protocols and Delegates**

We discussed that Swift does not support multiple inheritance, instead it uses protocols and delegation for similar functionality.

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 delegat*or*", 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. You specify what tasks you want your delegate to handle and the delegate has to agree to do those, specific implementation is up to the delegate.

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.
- For a class to act as a delegate it needs to conform to a protocol

Protocols

A protocol defines interfaces (signatures for methods) and rules for using them.
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 (slide)

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:

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 https://developer.apple.com/documentation/uikit/uitextfield
    - A text field is a single line area for entering text. https://developer.apple.com/design/human-interface-guidelines/ios/controls/text-fields/
    - UITextFieldDelegate protocol https://developer.apple.com/documentation/uikit/uitextfielddelegate
    - 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 https://developer.apple.com/documentation/uikit/uialertcontroller https://developer.apple.com/design/human-interface-guidelines/ios/views/alerts/
    - 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. https://developer.apple.com/design/human-interface-guidelines/ios/views/action-sheets/
    - 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 present(_:animated:completion:) to present your alert or action sheet

**Tip Calculator**
(tipcalculator)
Create a single view application named tipcalculator.

TextFields
Go into the storyboard and add 2 text fields with labels next to them(check amount and tip %),
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 placeholder values if you want. Placeholder text acts different than a value for text. When the app runs, placeholder text is a light grey and when the user starts typing it is automatically overwritten. If you just put in a default value for text then the user needs to backspace over it to put in their own value. It's more user-friendly to use placeholder text as an indicator of what should be in that text field.
You can also set the text fields to clear when editing begins(check box).
You can also customize the return key to say Done if you want.

Connect the text fields as outlets checkAmount and tipPercent.
Now these variables are defined in our swift file.

Run it and 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.

If the keyboard doesn't appear, the simulator may be configured to work as if a hardware keyboard had been connected. To fix that, uncheck Hardware | Keyboard | Connect Hardware Keyboard in the simulator and try again.

Keyboard

We need to adopt the UITextField protocol.

```
class ViewController: UIViewController, UITextFieldDelegate
```

Implement the UITextFieldDelegate method that is called when the return/done button is tapped

```
    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
        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 7.

Stepper

Add a label and stepper for the number of people. In the attributes inspector you can select min and max values, current value, and step which is the number it increases/decreases by. Make the initial value 1, min 0(for demo purposes later), max maybe 20, and step 1.
Since the initial value is 1, make its label say "1 person". We'll want this label to reflect the value of the stepper so the user knows what the value is.
What connections do you think we need?
An outlet for the label called peopleLabel.
An outlet for the stepper called peopleStepper
An action for the stepper called updatePeople

In the swift file look at the UIStepper class.
How do we get access to its value? What type is the value?
Let's implement updatePeople so the label shows the value of the stepper.

```
    @IBAction func updatePeople(_ sender: UIStepper) {
        if peopleStepper.value == 1 {
            peopleLabel.text = "1 person"
        } else {
            peopleLabel.text = String(format: "%.0f", peopleStepper.value +
" people"
        }
    }
```

Calculations

Back in the storyboard add 3 labels(tip, total, total per person) with labels next to those to match mine. Connect the tip, total and total per person labels as outlets tipDue, totalDue, and totalDuePerPerson.

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
        }

        let numberOfPeople = peopleStepper.value
        let tip=amount*pct
        let total=amount+tip
        var personTotal : Float = 0.0 //specify Float so it's not a Double
        if numberOfPeople > 0 {
            personTotal = total / Float(numberOfPeople)
        }

        //format results as currency
        let currencyFormatter = NumberFormatter()
        currencyFormatter.numberStyle=NumberFormatter.Style.currency //set
the number style
        tipDue.text=currencyFormatter.string(from: NSNumber(value: tip))
//returns a formatted string
        totalDue.text=currencyFormatter.string(from: NSNumber(value: total))
        totalDuePerPerson.text=currencyFormatter.string(from:
NSNumber(value: personTotal))
    }
```

Look at the NumberFormatter class to understand how we're using it.
When should we call updateTipTotals()?

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.

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

We should also update the totals when the number of people changes.
Update updatePeople to call `updateTipTotals()`

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)

```
else {
            //create a UIAlertController object
            let alert=UIAlertController(title: "Warning", message: "The
number of people must be greater than 0", preferredStyle:
UIAlertController.Style.alert)
            //create a UIAlertAction object for the button
            let cancelAction=UIAlertAction(title: "Cancel",
style:UIAlertAction.Style.cancel, handler: nil)
            alert.addAction(cancelAction) //adds the alert action to the
alert object
            let okAction=UIAlertAction(title: "OK", style:
UIAlertAction.Style.default, handler: {action in
                self.peopleStepper.value = 1
                self.peopleLabel.text? = "1 person"
                self.updateTipTotals()
            })
            alert.addAction(okAction)
            present(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.

Layout
Using stack views this layout works well.
For each label and it's corresponding textfield/label select them and embed in a horizontal stack view.
For the empty labels it might help to put some text in them to easily find them.

Set distribution to fill proportionally.
Don't worry about any other spacing yet or the layout errors you get.

> For the vertical spacing select all these stack views and embed them in another stack view, vertical axis this time. Set the alignment to fill and distribution to equal spacing. Use the spacing value to space them out in the stack view.

Now we can position this stack view by adding constraints to the superview in all directions – leading, trailing, top, and bottom (20 for each so there's a little room).
The beauty of this is since the stack view is pinned in all directions when you rotate to landscape orientation the constraints to the edges stay the same and the equal spacing inside the stack view adjusts to make up for less vertical space.

Scroll View

Often apps simple don't have enough vertical room and you want the user to be able to scroll the view.
Select the View Controller and in the size inspector change simulated size to Freeform and give the height 700.
Embed stack view in View (Editor | Embed In | View).
Embed View in Scroll View (Editor | Embed In | Scroll View).
(You don't have to use embed, you can always get them from the object hierarchy and drag them over. Just make sure they get added in the correct hierarchy)
Select the scroll view and add 4 constraints – leading, trailing, top, and bottom with the value 0. This ensures the scroll view takes up the entire view regardless of device size.
With the scroll view still selected go into the size inspector and change the height to be 700.
We have our stack view embedded in a View because the scroll view needs to have just 1 view in it, and that holds everything else.
Select the view that's in the stack view and make sure it will fill up the screen like the scroll view by adding the same constraints.
With the view still selected go into the size inspector and change the height to be 700.
In the document hierarchy control+click and drag from the embedded view to the scroll view and chose equal widths. This sets a constraint that ensures they will always be the same width.
Now let's position the stack view. Select the stack view and delete any constraints that it has.
Then add leading, trailing, top, and bottom constraints of 20 (not to the margins).
Now because we want to try scrolling, in the attributes inspector change the spacing for the stack view to 80. This will space out the embedded stack views and the user will need to scroll. Or try it in landscape orientation where you'll need to scroll.