EE443 / EE593 Mobile Application Development

Lily Osorno

Chapter 13: Delegates and Protocols

Lily Osorno

Where we are: An add item screen that lets you enter text… but adds nothing.

Where we want to go: Turn that text into a checklistItem object and add it to the items array!

How?

The Add Item screen needs to communicate with the Checklist View Controller. We'll go over 2 ways to do this:

1. The "messy" way
2. The delegate way

Hollomans, M. Farook, F. *iOS Apprentice.* (2017)

The Messy Way

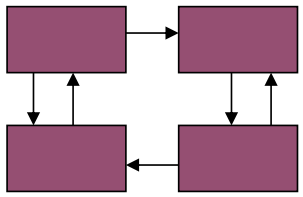
```
class AddItemViewController: UITableViewController, . . . {
   // This variable refers to the other view controller
   var checklistViewController: ChecklistViewController

   @IBAction func done() {
      // Create the new checklist item object
      let item = ChecklistItem()
      item.text = textField.text!

      // Directly call a method from ChecklistViewController
      checklistViewController.add(item)
   }
}
```

- AddItemViewController creates and accesses ChecklistViewController as a variable.
- The done button **directly** creates and adds a new ChecklistItem to ChecklistViewController.
- Meaning: AddItemViewController gets full range of access to all of ChecklistViewController's variables and functions

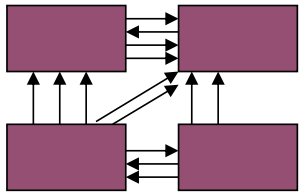
Data Coupling – Software Design

- Loose Coupling: Very little data other than what's necessary is passed. **Good** programming practice!

- Tight Coupling (the messy way): Here, content coupling -- the called function has full access to all the data and features of the calling function. **Bad** programming practice!

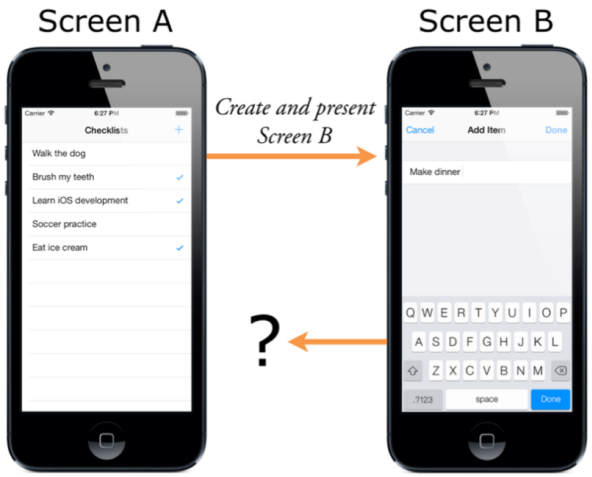


Loosely coupled
some dependencies



Highly coupled
many dependencies

Downsides?



Screen A knows all about screen B, but B knows nothing of A

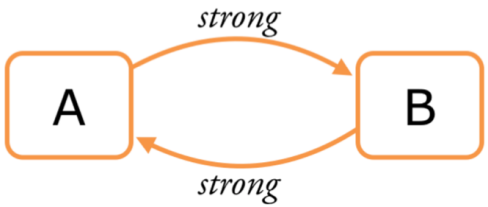
Hollomans, M. Farook, F. *iOS Apprentice.* (2017)

1. Bad coding practice: For security, you don't want a function that is called to know too much about who called it.

2. The add item screen will always take you directly back to the main screen… so directly linking the two screens directly means you can't call it from anywhere else in the app!

Bonus: Weak vs. Strong Variables

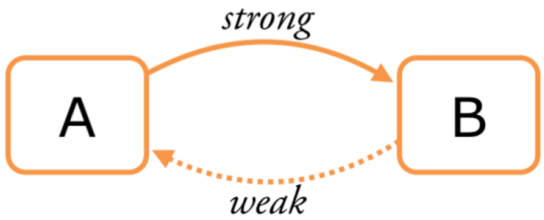
- Relationships between objects can be weak or strong
- An object is destroyed/the memory deallocated when there are no more strong references to it
- If two objects both have a strong references to each other, this creates an **ownership cycle**



Ownership cycle

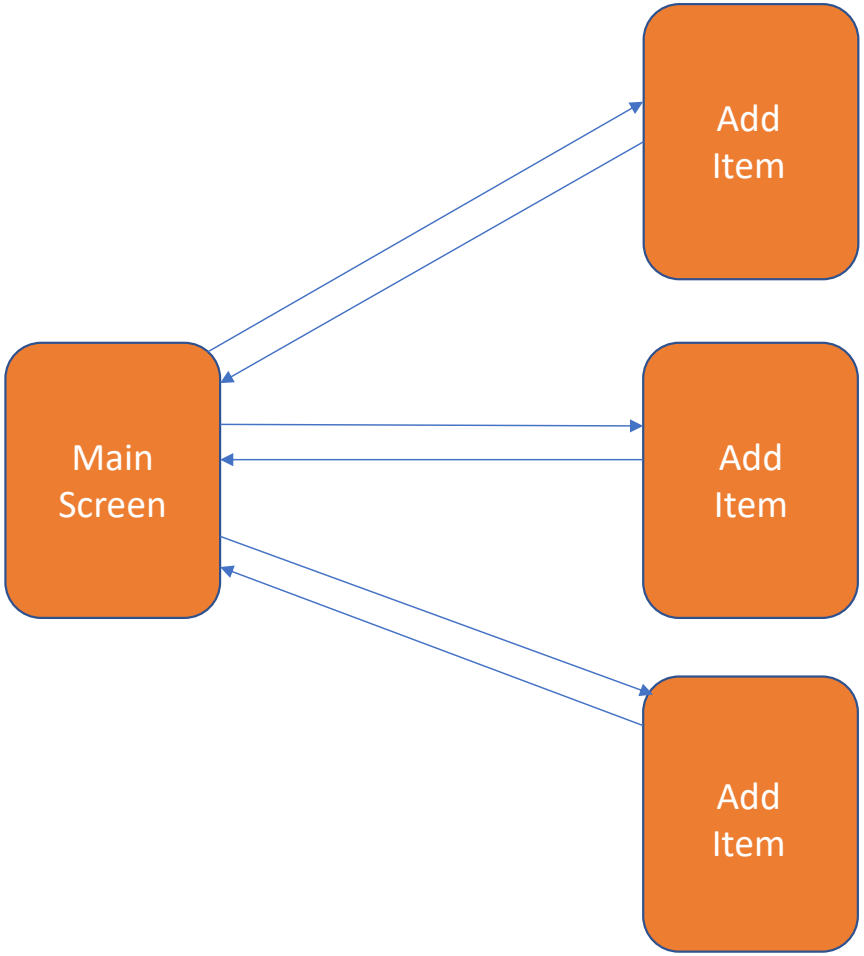
- If the memory cannot be deallocated, this creates the potential for a **memory leak**: when an object should be destroyed but isn't.

-> this may cause your app to crash!

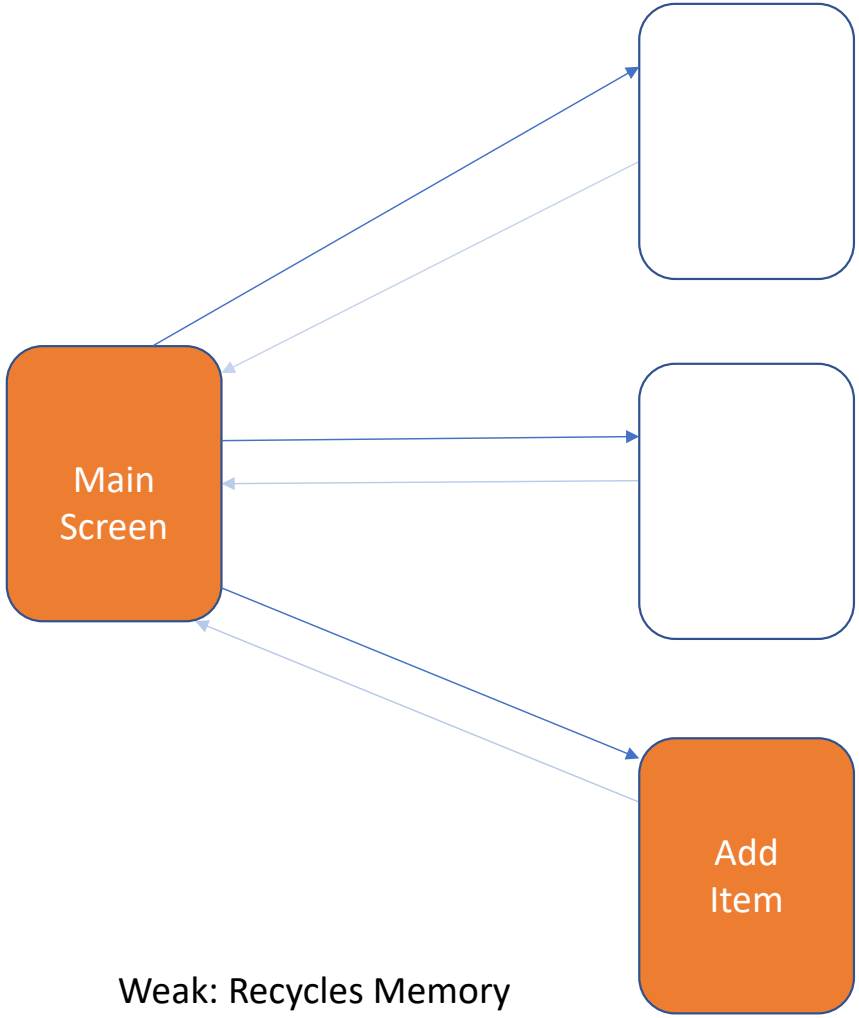


"healthy" relationship

Note: We also declare @IBOutlets using the keyword weak. This is not to avoid an ownership cycle, but to make sure that the view controller isn't the owner of any views from outlets. ----- The ownership cycle is most common among delegates.



Strong: Ownership Cycle



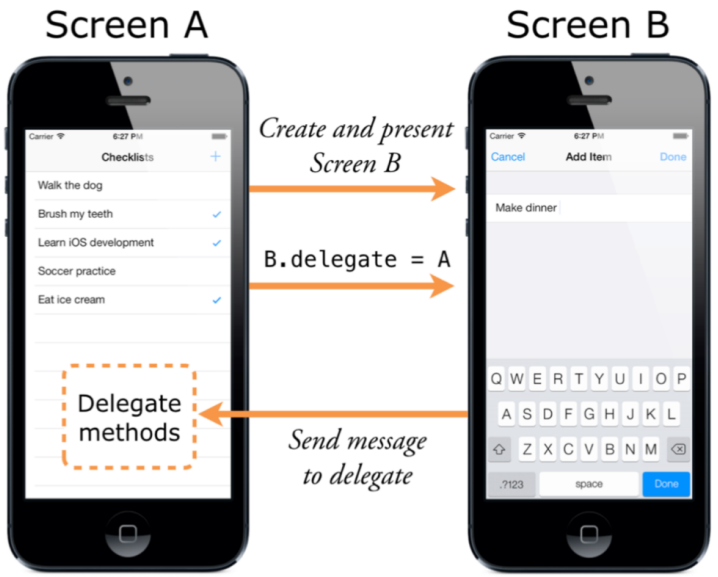
Weak: Recycles Memory

If screen B can't know about screen A…

How do we send information back?
Simple:

The delegate!

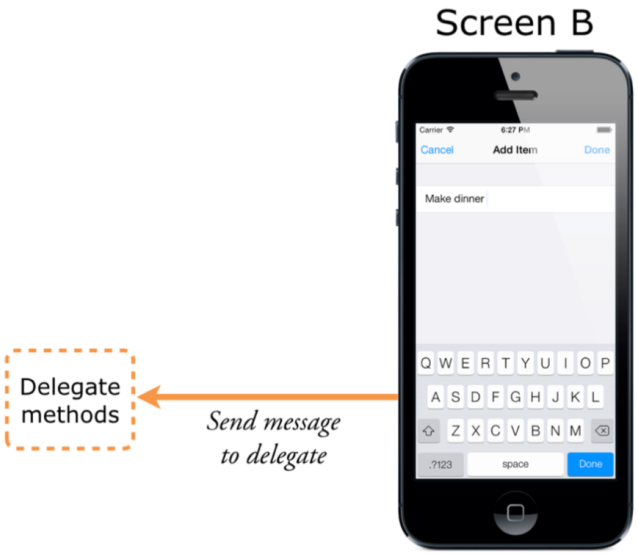
We've seen them before: tableview (responds to taps on rows) and the text field (checks text length).



Screen A launches screen B and becomes its delegate

Hollomans, M. Farook, F. *iOS Apprentice.* (2017)

The Delegate



This is what Screen B sees: only the delegate part, not the rest of screen A

Hollomans, M. Farook, F. *iOS Apprentice.* (2017)

Screen B only sees and only talks to the delegate, it doesn't care or know anything about who's on the other side.

Example: Germany sends a delegate to the U.S.

The Delegate Protocol

Example: Another Protocol

```
protocol AddItemViewControllerDelegate: class {
   func addItemViewControllerDidCancel( _ controller:
            AddItemViewController)

   func addItemViewController(_ controller:
            AddItemViewController,
            didFinishAdding item: ChecklistItem)
}
```

- Add after the import line, but before the start of the class ☺
- First Method: addItemViewControllerDidCancel (_:) for when user presses cancel
- Second Method: addItemViewController(_:didFinishAdding:) for when user presses done

After we've filled these in, we can call the protocol instead of directly calling the ChecklistViewController! #saftey

The Delegate Protocol

- Let's use the delegate! Declare a delegate variable in AddItemViewController.

```
weak var delegate: AddItemViewControllerDelegate?
```

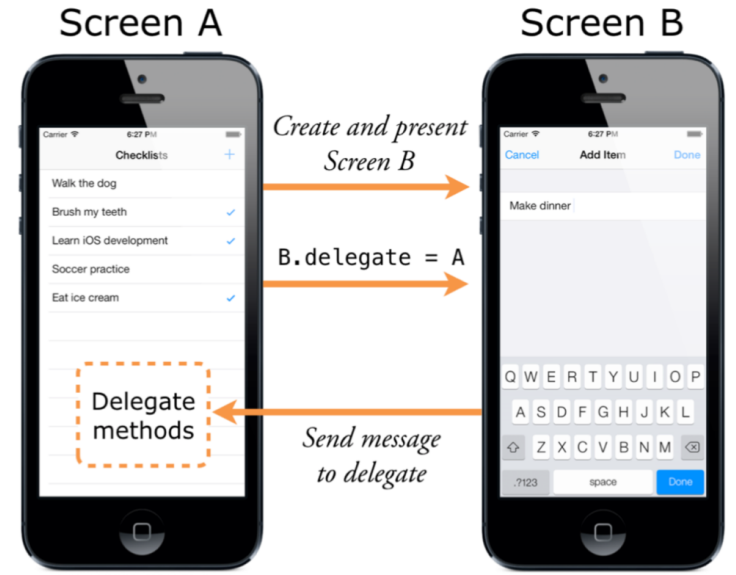
- Next, fill in the done and cancel functions so that they tell the delegate they were pressed.

```
@IBAction func cancel() {
    delegate?.addItemViewControllerDidCancel(self)
}

@IBAction func done() {
    let item = ChecklistItem()
    item.text = textField.text!
    item.checked = false

    delegate?.addItemViewController(self, didFinishAdding:
            item)
}
```

Surprise! They don't work

We haven't actually told the delegate to do anything yet! So it knows what's happening on screen B, but doesn't yet know to tell screen A anything.

We still need to connect the delegate to the ChecklistViewController.



Screen A launches screen B and becomes its delegate

Hollomans, M. Farook, F. *iOS Apprentice.* (2017)

Adding the Delegate

- modify the class:

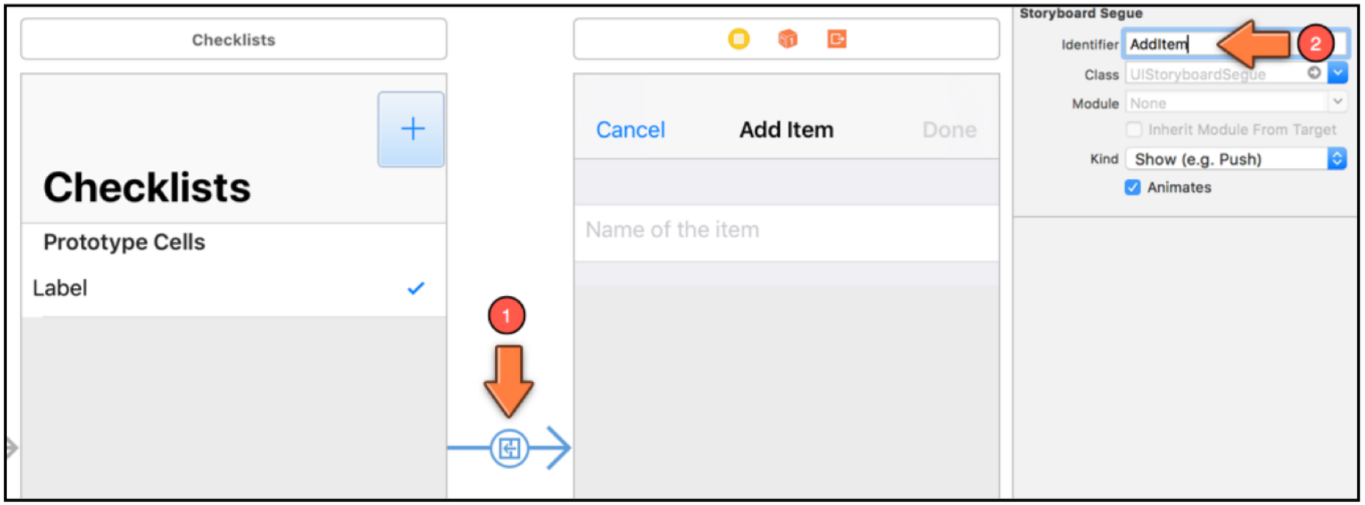
```
class ChecklistViewController: UITableViewController,
AddItemViewControllerDelegate {
```

- Add the protocol functions to ChecklistViewController:

```
func addItemViewControllerDidCancel( _ controller: AddItemViewController) {
   navigationController?.popViewController(animated:true)
}

func addItemViewController( _ controller: AddItemViewController, didFinishAdding
item: ChecklistItem) {
   navigationController?.popViewController(animated:true)
}
```

Name the segue

- Next, in the **Attributes Inspector** set the segue identifier to match what we told the delegate to call on.



Naming the segue between the Checklists scene and the Add Item scene

Hollomans, M. Farook, F. *iOS Apprentice.* (2017)

Adding the Call to Create the Delegate

Add the prepare-for-segue method, aka prepare(for:sender:) method.

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {

   // 1
   if segue.identifier == "AddItem" {
      // 2
      let controller = segue.destination as! AddItemViewController
      // 3
      controller.delegate = self
   }
}
```

1 - Make sure you're addressing the correct segue (there can be more than one segue per view controller)

2 – cast the segue to be of the type AddItemViewController

3 – set the delegate property to self so AddItemViewController knows that ChecklistViewController is the delegate

Et Voilà!

```
func addItemViewController(_ controller:
AddItemViewController,didFinishAdding item: ChecklistItem)
{
   let newRowIndex = items.count
   items.append(item)

   let indexPath = IndexPath(row: newRowIndex, section: 0)
   let indexPaths = [indexPath]
   tableView.insertRows(at: indexPaths, with: .automatic)
   navigationController?.popViewController(animated:true)
}
```

- Next, in ChecklistViewController remove the function addItem (as we now have a delegate to do so)
- Edit didFinishAdding to add the following code: (it's the same as before, only now you don't create the new ChecklistObject, you tell AddItemViewController to do that. ☺

Questions?