```swift
//
//  DelegateCheatSheet.swift
//  Delegate Lab Cheat Sheet
//
```

DELEGATE CHEAT SHEET - iOS/Swift

WHAT IS A DELEGATE?
A design pattern where one object (delegator) hands off tasks to another object (delegate).
Like a BOSS-ASSISTANT relationship:
• BOSS (delegator): Knows WHAT needs to be done
• ASSISTANT (delegate): Knows HOW to do it
• PROTOCOL: The job description both agree on

WHY USE DELEGATES?
• LOOSE COUPLING: Objects don't need to know about each other
• SEPARATION OF CONCERNS: Each class handles its own responsibilities
• REUSABILITY: Same delegator can work with different delegates

HOW TO CREATE A CUSTOM DELEGATE (5 STEPS)

```swift
// STEP 1: Define the Protocol
protocol TaskManagerDelegate: AnyObject {
   func taskDidStart(_ task: String)
   func taskDidFinish(_ task: String, success: Bool)
}

// STEP 2: Create Delegator Class
class TaskManager {
   // STEP 4: Add delegate property (ALWAYS WEAK!)
   weak var delegate: TaskManagerDelegate?

   func performTask(_ task: String) {
      // Notify delegate when events happen
      delegate?.taskDidStart(task)
      // Do work...
      delegate?.taskDidFinish(task, success: true)
   }
}

// STEP 3: Create Delegate Class
```

```
class ViewController: UIViewController, TaskManagerDelegate {
    let taskManager = TaskManager()

    override func viewDidLoad() {
        super.viewDidLoad()
        // STEP 5: Assign delegate property
        taskManager.delegate = self
        taskManager.performTask("Download Data")
    }

    // Implement protocol methods
    func taskDidStart(_ task: String) {
        print("Started: \(task)")
        // Update UI to show loading state
    }

    func taskDidFinish(_ task: String, success: Bool) {
        print("Finished: \(task) - Success: \(success)")
        // Update UI to show result
    }
}
```

COMMON UIKit DELEGATES

```
// UITableViewDelegate
class ViewController: UIViewController, UITableViewDelegate {
    @IBOutlet weak var tableView: UITableView!

    override func viewDidLoad() {
        super.viewDidLoad()
        tableView.delegate = self
    }

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        print("Selected row: \(indexPath.row)")
    }

    func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat
{
        return 60.0
    }
}

// UITextFieldDelegate
```

```
class ViewController: UIViewController, UITextFieldDelegate {
    @IBOutlet weak var textField: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
        textField.delegate = self
    }

    func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        textField.resignFirstResponder() // Dismiss keyboard
        return true
    }

    func textField(_ textField: UITextField, shouldChangeCharactersIn range: NSRange,
                   replacementString string: String) -> Bool {
        // Only allow numeric input
        return CharacterSet.decimalDigits.isSuperset(of: CharacterSet(charactersIn: string))
    }
}
```

SWIFTUI ALTERNATIVES

```
// Closures (Most Common Replacement)
struct TaskView: View {
    let onTaskStart: (String) -> Void
    let onTaskFinish: (String, Bool) -> Void

    var body: some View {
        Button("Start Task") {
            onTaskStart("Processing")
            // Later...
            onTaskFinish("Processing", true)
        }
    }
}

// @Binding for Two-Way Communication
struct ToggleView: View {
    @Binding var isOn: Bool  // Read/write access to parent's state

    var body: some View {
        Toggle("Switch", isOn: $isOn)
    }
}
```

```
// Usage in Parent View
struct ParentView: View {
    @State private var toggleState = false

    var body: some View {
        VStack {
            Text("State: \(toggleState.description)")
            ToggleView(isOn: $toggleState)
        }
    }
}
```

CRITICAL RULES & BEST PRACTICES

```
// ⚠️ PREVENT RETAIN CYCLES - ALWAYS USE WEAK!
weak var delegate: MyDelegate?  // ✅ CORRECT
var delegate: MyDelegate?       // ❌ WRONG - causes memory leak!

// Optional Method Calls - Always use safe calling
delegate?.methodName()        // ✅ Safe optional chaining
delegate!.methodName()        // ❌ Unsafe - could crash

// Protocol Inheritance - Use AnyObject for class-only protocols
protocol MyDelegate: AnyObject {
    func didUpdateData()
}

// Optional Methods - Provide default implementation
extension MyDelegate {
    func optionalMethod() { }   // Empty default implementation
}
```

QUICK REFERENCE GUIDE

```
STEP 1: Define Protocol    → protocol MyDelegate: AnyObject { func somethingHappened() }
STEP 2: Create Delegator   → class Sender { weak var delegate: MyDelegate? }
STEP 3: Implement Delegate → class Receiver: MyDelegate { func somethingHappened() { } }
STEP 4: Set Delegate       → sender.delegate = self
STEP 5: Call Methods       → delegate?.somethingHappened()
```

KEY POINTS TO REMEMBER

• Delegates enable communication between objects while keeping them separate

- Always declare delegate properties as WEAK to prevent memory leaks
- Use AnyObject protocol inheritance to ensure only classes can conform
- Check if delegate is nil before calling methods with optional chaining: delegate?.method()
- In SwiftUI, prefer closures and @Binding over custom delegates
- Common UIKit components (UITableView, UITextField) heavily use delegates

DELEGATES = CLEAN SEPARATION + FLEXIBLE COMMUNICATION