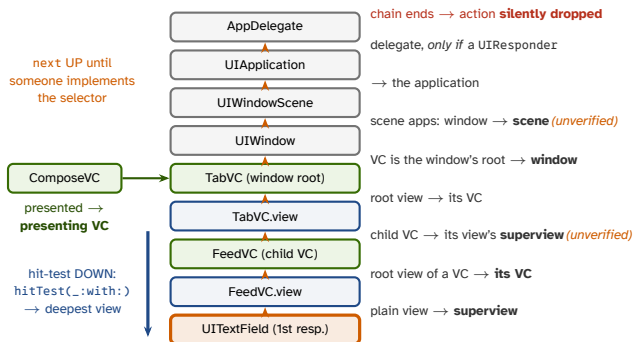


**In one line:** You use GoF patterns daily: the **responder chain** is Chain of Responsibility, **target-action/UndoManager** is Command (+ Memento), NotificationCenter/KVO/Combine are Observer, **class clusters** are Abstract Factory, appearance() is a Proxy, a ViewModifier a Decorator. Senior answer = **pattern + API + mechanism + trade-off**.

| Pattern                 | Where Apple put it                                                                       | Mechanism                                                                                       | Say it like this                                          |
|-------------------------|------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|-----------------------------------------------------------|
| <b>Chain of Resp.</b>   | responder chain: UIResponder.next; sendAction(_to: nil, from:for:)                       | selector not implemented? pass to next until someone does, or the chain ends                    | <i>Copy/Paste find a handler by walking the chain.</i>    |
| <b>Command</b>          | target-action, UIAction, UIKeyCommand, ObjC                                              | request stored as data, run later; the undo closure captures the <i>old value</i> = the memento | <i>UndoManager is a command stack over snapshots.</i>     |
| <b>+Memento</b>         | NSNotification; UndoManager                                                              | notify subscribers you don't know; @Observable tracks the properties a body <i>read</i>         | <i>broadcast → KVO → typed streams → access tracking</i>  |
| <b>Observer</b>         | NotificationCenter · KVO · Combine @Published · @Observable (iOS 17)                     | weak helper called at fixed hooks; a layout object = a whole algorithm, swapped at runtime      | <i>Delegate: many hooks, one owner. Layout: Strategy.</i> |
| <b>Delegation</b>       | tableView delegate + data source;                                                        | public abstract class; init returns a <i>private</i> subclass (___NSArrayI, ___NSArrayM)        | <i>Apple documents clusters as Abstract Factory.</i>      |
| <b>+Strategy</b>        | UICollectionViewLayout; keyDecodingStrategy                                              | proxy records setters, applies them when a view <b>enters a window</b>                          | <i>Not the bar — a recorder.</i>                          |
| <b>Abstract Factory</b> | <b>class clusters:</b> NSArray, NSNumber, NSString, NSDictionary                         | each call wraps: ModifiedContent<Content, Modifier>, nested                                     | <i>The wrapping is in the type — order matters.</i>       |
| <b>Proxy</b>            | UIAppearance; UINavigationController.appearance(); ObjC NSProxy                          | leaf + container share one interface; operations recurse                                        | <i>Hit-test walks it down, the chain up.</i>              |
| <b>Decorator</b>        | SwiftUI ViewModifier, .padding(), .background()                                          | makeUIView/updateUIView; its Coordinator adapts UIKit delegates                                 | <i>Imperative view behind a declarative protocol.</i>     |
| <b>Composite</b>        | UIView/CALayer trees, SwiftUI view tree, UIMenu(children:)                               | makeIterator(), then next() until nil                                                           | <i>for-in is sugar for it.</i>                            |
| <b>Adapter</b>          | UIViewRepresentable (UIKit→SwiftUI), UIHostingController (SwiftUI→UIKit)                 | hop 1 dispatches on the <i>value's</i> type, hop 2 on the <i>encoder's</i> (= format)           | <i>Double dispatch: one encode(to:), any for-format.</i>  |
| <b>Iterator</b>         | Sequence + IteratorProtocol, AsyncSequence                                               | first class whose canInit(with:) says true loads it (registered: <b>reverse</b> order)          | <i>How I stub the network in tests.</i>                   |
| <b>Visitor-like</b>     | Codable: JSONEncoder().encode(v) → v.encode(to:) → c.encode(x, forKey:)                  | copy a configured instance; Foundation copies are <b>shallow</b>                                | <i>Swift: var b = a.</i>                                  |
| <b>Chain→Strategy</b>   | URLProtocol; config.protocolClasses; URLProtocol.registerClass                           | only UIApplication is one per process (UIApplicationMain); others you may create                | <i>Shared instance ≠ singleton; I still inject it.</i>    |
| <b>Prototype</b>        | NSCopying: copy(), mutableCopy()                                                         |                                                                                                 |                                                           |
| <b>Singleton</b>        | UIApplication.shared vs shared <i>instances</i> (URLSession.shared, FileManager.default) |                                                                                                 |                                                           |

**Also (other sheets):** Template Method = VC lifecycle, layoutSubviews · Factory Method = class var layerClass · Facade = URLSession · Flyweight = UIImage(named:) · Interpreter = NSPredicate · Object Pool = cell reuse.

## Picture — the responder chain, exact rules



Touches start at the hit-tested view; keys, menus and nil-target actions at the **first responder**. Scene step: Apple forum answer — the article still says window → app.

## Interview traps

- ! Hit-test (down) ≠ responder chain (up): two walks over one Composite.
- ! A nil-target action nobody implements does **nothing** — no crash, no log.
- ! appearance() changed while a view is on screen: no effect until it re-enters a window.
- ! type(of: NSArray()) is a private subclass — never compare a cluster's class.
- ! URLSession.shared is a shared *instance*, not a singleton.

## Remember

**Chain walks up · Command stores the request · Observer fans out · Delegate answers back · Cluster hides the subclass · Proxy records · Modifier wraps · Codable dispatches twice.**

## Example — drive the patterns yourself

```
// Chain: nil target -> first responder -> up ("hide keyboard")
UIApplication.shared.sendAction(#selector(UIResponder
.resignFirstResponder), to: nil, from: nil)
// Chain -> Strategy: URLProtocol stubs the network in tests
final class StubProtocol: URLProtocol {
    nonisolated(unsafe) static var body = Data()
    override class func canInit(with r: URLRequest) -> Bool { true }
    override class func canonicalRequest(for r: URLRequest)
        -> URLRequest { r }
    override func startLoading() {
        client?.urlProtocol(self, didLoad: Self.body)
        client?.urlProtocolDidFinishLoading(self)
    }
    override func stopLoading() {}
}
let cfg = URLSessionConfiguration.ephemeral
cfg.protocolClasses = [StubProtocol.self]
// Command + Memento: the closure captures the OLD value
@MainActor func rename(_ doc: Doc, to new: String,
    undo: UndoManager) {
    let old = doc.title // the memento
    undo.registerUndo(withTarget: doc) { d in // use d, not doc
        rename(d, to: old, undo: undo) } // = registers redo
    undo.setActionName("Rename"); doc.title = new }
```

Doc: a final class with var title. Registering *inside* the undo handler is what makes **redo** work — during undo, UndoManager puts new registrations on the redo stack.

## Likely questions

1. A GoF pattern in UIKit? — responder chain = Chain of Responsibility.
2. How does Copy find its handler? — nil-target action up the chain.
3. Class cluster? — Abstract Factory: public abstract type, private concretes.
4. Decorator in SwiftUI? — ViewModifier → ModifiedContent.
5. Stub URLSession? — a URLProtocol in protocolClasses.
6. Undo/redo in UIKit? — UndoManager: a Command stack; closures capture the old value.
7. Is Codable a Visitor? — the same double dispatch: value type × encoder type.