

Recall drill — the named facts you missed

recall · drill

In one line: Cover the **blue**, say the answer *aloud*, uncover. Every line is a question you answered “I don’t know”, half-right, or backwards. **!** = missed twice / headline gap. Exact names, numbers, lists — the reasoning you already have.

Auto Layout

! UITableView — 4 properties → axis · distribution (sizing *along* axis) · alignment (across axis) · spacing

distribution values → `.fill` (default) · `.fillEqually` · `.fillProportionally` · `.equalSpacing` · `.equalCentering`

! Priority values → `.required` = **1000** · `.defaultHigh` = **750** · `.defaultLow` = **250**

Content hugging → resists *growing* past intrinsic size; default **250**

Compression resistance → resists *shrinking* below intrinsic size (truncation); default **750**

Who loses a tie? → the **lower** priority stretches/shrinks; nudge ± 1 (749, 251) to decide

#1 cause: code view at 0/wrong place → forgot `translatesAutoresizingMaskIntoConstraints = false`

Facts per view → *x*, *y*, width, height — size may come from **intrinsic content size**

Unsatisfiable constraints → ≥ 2 contradictory *required* (1000) → engine breaks one arbitrarily + log; optional (<1000) breaks silently

Self-sizing cells → `rowHeight` = `.automaticDimension` + non-zero `estimatedRowHeight` + unbroken top → bottom chain in `contentView`

Size classes → horizontal/vertical × `.compact` / `.regular`; iPhone portrait = compact W, regular H; iPad = regular/regular

Networking

! JSON user_name → `userName` → (1) enum `CodingKeys`: `String`, `CodingKey` { case `userName` = "user_name" } (per key) (2) `decoder.keyDecodingStrategy` = `.convertFromSnakeCase` (blanket) (3) custom `init(from:)`

! 3 error layers (Reach-Receive-Read) → `URLLError` (transport) → HTTP `statusCode` not in 200..`<300` (*not thrown*) → `DecodingError` (`.keyNotFound` / `.typeMismatch` / `.dataCorrupted`)

! dataTask completion thread → `URLSession's background` queue, never main → `DispatchQueue.main.async`; call `.resume()`

async/await data(from:) → resumes on the caller's actor — `@MainActor` caller needs no hop

Pinning delegate → `urlSession(.didReceive: completionHandler:)` with `URLAuthenticationChallenge` → `serverTrust` → `SecTrustEvaluateWithError` → SPKI hash → `.useCredential` / `.cancelAuthenticationChallenge`. Default TLS validation: **yes**

VC & app lifecycle

VC order → `loadView` → `viewDidLoad` (once) → `viewWillAppear` → `viewDidAppear` → `viewWillDisappear` → `viewDidDisappear`

Frame valid from → `viewWillLayoutSubviews` — *not* `viewDidLoad`

App states → Not Running → Inactive → Active ↔ Inactive → Background → Suspended

Memory & value semantics

! 3 memory tools → **Memory Graph Debugger** (cycles: who retains whom) · **Leaks** (unreachable blocks) · **Allocations** (growth; Mark Generation)

weak vs unowned → weak: Optional, auto-nils, target may die first · unowned: non-optional, target outlives self, crash if touched after dealloc

Does var x: C? break a cycle? → **No** — ? = nil-ability; weak/unowned = ownership

weak var delegate: P? needs → protocol P: AnyObject

! { [x] in print(x) }, then x = 20 → prints **10**: capture list = **by-value snapshot** at creation. No list → captures the variable → 20. [obj] copies the *pointer*

! Copy-on-write → Array Dictionary Set String (+Data): struct wrapping class storage; `isKnownUniquelyReferenced(&storage)` — 1 → mutate in place, >1 → copy first. Your structs: *no* COW for free

! mutating → lets a struct/enum method change self; caller must be var; classes never need it; on a let struct → compile error

When must it be a class? → identity/shared state · inheritance · ObjC interop · deinit

Unit-test a leak → `weak var ref`; create SUT in inner scope; after scope `XCTAssertNil(ref)`

SwiftUI

some View → opaque type: compiler *knows* the concrete type; hidden from the caller

@StateObject vs @ObservedObject → view creates & owns (once) vs passed in; `@ObservedObject var vm = VM()` resets on parent re-render

@Binding → two-way ref to a parent's @State, made with \$; owns nothing

iOS 17 @Observable → owned → @State; injected → plain property

Testing

! Test pyramid → Unit **70%** · Integration **20%** · UI **10%**; inverted = “ice-cream cone”

5 test doubles → **Dummy** (fills a slot) · **Stub** (canned answers) · **Spy** (stub + records calls) · **Mock** (pre-set expectations, self-verifies) · **Fake** (working simplified impl, e.g. in-memory)

#expect vs #require → soft (continues) vs hard (throws, stops)

XCTest per-test order → new instance → `setUpWithError` → `setUp` → test → `tearDown` blocks (LIFO) → `tearDownWithError` → `tearDown`

Architecture

Logic lives in (MVC / MVP / MVVM) → Controller / Presenter (View behind a **protocol**, no UIKit) / ViewModel (View **binds**, VM has no View ref)

Facade — what + risk → one simple interface over a subsystem (URLSession, Repository); risk: swells into a **god object** — delegate, don't implement

Adapter — textbook iOS → `UIViewRepresentable` / `UIHostingController` (UIKit ↔ SwiftUI)

Coordinator solves → a VC that pushes its next screen knows its siblings → not reusable; VC signals intent, Coordinator navigates

Repository read → cache → miss → API → write back → return; caller never knows the source

DI vs DIP → DI = technique (pass deps in); DIP = principle (depend on abstractions)

Design patterns

! Singleton downsides → hidden dependencies · hard to test / state leaks · global mutable state · tight coupling (breaks DIP). Impl: `static let shared` (lazy, thread-safe) + `private init`

! Strategy vs State → Strategy: **client picks** the algorithm (*stranger*) · State: object **transitions itself** (*self*)

! Observer's problem → **leaks** / retain cycles if not unsubscribed → `AnyCancellable`, `store(in:)`, `removeObserver`

! Command → **undo** → `execute()` + `undo()` (captures state); undo stack: `undo` = `pop`, `undo()`, `push` to redo

! Proxy variants → **Virtual** · **Protection** · **Remote** · **Caching** (“Very Private Rooms Cache”); lazy `var` = virtual proxy

Interface: Adapter / Decorator / Facade / Proxy → changes it / keeps + adds behaviour / new simpler one / keeps + gates access

! Template Method → **Hollywood principle**: “don't call us, we'll call you” — base calls your hooks; e.g. `UIViewController` lifecycle

! Responder chain direction → **up**: first responder → `superview` → VC → window → `UIApplication` → app delegate. **Hit-testing** goes **down**

Mediator → peers talk via a hub: **N²** → **N** links (Coordinator, VM, `UINavigationController`)

Factory Method vs Abstract Factory → FM: subclass overrides to pick **one** product · AF: one object makes a **family**. `create(type:)` switch = Simple Factory

! Decorator vs subclass → N features → **2^N** subclasses vs N stackable wrappers; Swift: `ViewModifier`

Builder → fixes telescoping init; Swift: default params; recipe object = **Director**

Iterator in Swift → `IteratorProtocol.next()` → `Element?` (nil = end) + `Sequence.makeIterator()`

Strategy serves → **Open/Closed + Dependency Inversion** (not ISP)

SOLID — exact one-liners

S → one, and only one, **reason to change**

O → open for extension, closed for modification

L → subtypes substitutable for their base types (Square:Rectangle breaks it)

! I → **clients** should not be forced to depend on methods they do not use

! D → high- *and* low-level modules **both** depend on abstractions; abstractions don't depend on details

Memory hooks — say them once a day

Errors → **Reach, Receive, Read**

Tools → **Graph** = cycles · **Leaks** = unreachable · **Allocations** = growing

Priorities → 1000/750/250, spaced by 250 so ± 1 breaks ties; 750 = “don't shrink me”, 250 = “don't stretch me”

Capture list → a **photo**: values freeze; references freeze the *pointer*

Ownership → weak/unowned change ownership; ? only changes nil-ability

Stack view → axis = direction · distribution = *along* · alignment = *across* · spacing = gaps

Adapter · Decorator · Proxy → *changes* · *adds* · *gates* (a bouncer)

Related: auto-layout · urlsession-networking · arc-ownership · value-vs-reference · app-and-vc-lifecycle · swiftui-state · gof-creational-structural · gof-behavioural · solid · testing-fundamentals