

In one line: XCTest makes a **fresh XCTestCase instance per test method** and wraps each one in `setUp` → `test` → `tearDown`; Swift Testing (import Testing, Xcode 16+) does the same with a `@Suite`'s `init/deinit`, `@Test` and `#expect/#require`. The leak test = **weak ref** → **drop the strong** → **assert nil**.

How it works

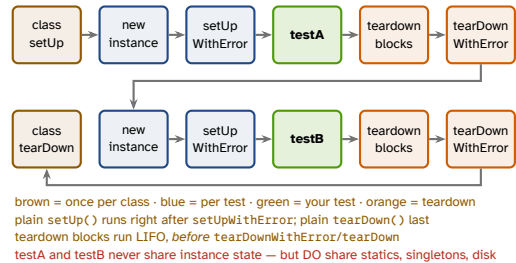
- **Discovery:** subclass XCTestCase; every instance method named `test...`, no args, returns Void (may be throws/async) runs. Order is *not* a contract — tests must be independent.
- **Lifecycle:** class func `setUp()` **once** per class → for each test: `new instance` → `setUpWithError()` → `setUp()` → `test...` → `addTearDownBlocks (LIFO)` → `tearDownWithError()` → `tearDown()` → ... class func `tearDown()` once.
- Prefer the `WithError` variants: they throws, so setup can try and a throw fails the test cleanly (no try!).
- Instance state is fresh, but **statics, singletons, UserDefaults, files** are not — and the instance lives on in the runner: **nil the SUT in tearDown**.
- **Asserts:** `XCTAssertEqual/NotEqual · True/False · Nil/NotNil · Identical(==) · GreaterThan...` · `XCTAssertThrowsError(expr){ err in ... }` · `XCTAssertNoThrow · XCTFail("msg")` (unconditional) · `try XCTUnwrap(opt)` (fail+stop, not crash) · `throw XCTSkip / XCTSkipIf`. All take message, capture file/line.
- **Performance:** `measure { }` runs the block several times, compares the average to a stored **baseline** (per device+OS). `measure(metrics:)` with `XCTClockMetric · XCTCPUMetric · XCTMemoryMetric · XCTStorageMetric · XCTOSSignpostMetric`. Baselines don't transfer between machines.
- **Test plan** (`.xctestplan`): configurations (locale, env, sanitizers), random order, parallel, repetitions, coverage.

Example — the leak test (you were PARTIAL)

```
func test_viewModel_isDeallocated() {
    weak var weakRef: FeedViewModel?
    do {
        let sut = FeedViewModel(api: StubAPI())
        weakRef = sut // inner scope = release point
        sut.start() // observe WITHOUT retaining
        sut.start() // code that may cycle
        // last strong ref gone here
    }
    XCTAssertNil(weakRef, "leak: retain cycle keeps VM alive")
}

// reusable form — run AFTER the test body, for every makeSUT():
func trackForLeaks(_ o: AnyObject, line: UInt = #line) {
    addTearDownBlock { [weak o] in
        XCTAssertNil(o, "leaked", line: line) }
}
```

Picture — one test class, two tests



XCTest vs Swift Testing

	XCTest	Swift Testing
declare	func test...() in XCTestCase	@Test func any() (free or in type)
group	the class	@Suite struct (fresh per test)
setup/teardown	setUp/tearDown	init / deinit
soft check	XCTAssertEqual(a,b)	#expect(a == b) — shows both values
hard check	try XCTUnwrap(x)	try #require(x) (throws, stops)
errors	XCTAssertThrowsError	#expect(throws: E.self){...}
async wait	XCTestExpectation	await confirmation{...}
table tests	loop (one result)	@Test(arguments: [...]) — 1 case each
traits	—	.disabled .enabled(if:) .tags .timeLimit .bug .serialized
parallel	opt-in, sim clones	default , in-process
UI / perf	XCUITest, measure	not supported — stay in XCTest

Interview traps

- ! `deinit{print}` and Instruments are *debugging*, not a **test** — the automated check is `weak-ref-then-XCTAssertNil`.
- ! `trackForLeaks` on a SUT stored in `var sut` **always fails**: `tearDown` blocks run *before* `tearDown` nils it. Create it in a local `makeSUT()`.
- ! `XCTUnwrap` vs `!`: a force-unwrap crash kills the *whole runner*; `XCTUnwrap` fails one test.
- ! Porting `XCTAssertEqual(a,b)` as `#expect(a,b)` — idiomatic is `#expect(a == b)`. Both frameworks coexist in one target.
- ! `measure` on shared CI flakes — baselines are per device+OS.

Remember

Weak → **scope** → **nil**. `#expect` continues, `#require` stops. New instance per test; `tearDown` blocks LIFO before `tearDown`.

Likely questions

1. Order of `setUp` variants? — class `setUp` once; per test `setUpWithError` → `setUp` → `test` → `blocks` → `tearDown(WithError)`.
2. Test a VM for retain cycles? — `weak var ref`, drop strong in scope, `XCTAssertNil`.
3. `#expect` vs `#require`? — soft (records, continues) vs hard (throws).
4. Why parameterized over a loop? — each argument reported and re-runnable alone.
5. Measure memory in a test? — `measure(metrics: [XCTMemoryMetric()])`.