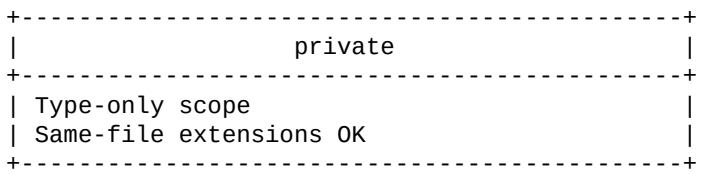
PRIVATE

Definition:

Restricts access to the enclosing type and its same-file extensions.

Scope Diagram:



Code Example:

```
// File: Vault.swift
class BankVault {
    private var code = 9876            // Only BankVault & its same-file extensions can acces
    func reveal() -> Int { return code }
}
extension BankVault {
    func updateCode(to newCode: Int) { code = newCode } // allowed
}
class SecurityGuard {
    func inspect() {
        let v = BankVault()
        // print(v.code) // ■ Error: 'code' is private to BankVault
        print(v.reveal()) // ■ allowed
    }
}
```

Practice Questions (with answers):

Q: What does 'private' restrict access to?
A: It restricts access to the enclosing declaration (the type) and its same-file extensions.

Q: Can another class in same file access a private property?
A: No — only the declaring type and its same-file extensions can access it.

Q: Can an extension in same file access private members?
A: Yes, extensions of the same type in the same file are considered part of the type.

Q: Is private visible across files?
A: No, private is not visible across files unless accessed through the type's public/internal interfaces.

Q: Should private be default for helper methods?
A: Yes — use private for helper methods that shouldn't be part of the type's API.

Q: Does private prevent unit tests from accessing members with @testable?
A: Yes — @testable does not grant access to private members.

Q: Can nested types access the private members of enclosing type?
A: Yes, nested types declared inside the same type can access private members of the enclosing type.

Q: Does Swift allow fileprivate and private together?
A: They are different keywords; you choose one — private is narrower than fileprivate.

Q: Will marking a stored property private stop extensions in other files?
A: Yes — only same-file extensions of the type can access private members.

Q: Is a private initializer usable outside the type?
A: No — a private initializer cannot be called outside the declaring type (except same-file extensions).

Q: Can protocols have private members?
A: Protocols cannot have private requirements; conforming types implement the requirements with appropriate access.

Q: Does private affect memory layout or performance?
A: No — access control is compile-time only and doesn't change runtime memory layout.

Q: Can a subclass access its parent's private members?
A: No — subclasses cannot access private members of their superclass unless via same-file extension context.

Q: Why use private instead of fileprivate?
A: Prefer private for tighter encapsulation; use fileprivate only when multiple types in same file must share.

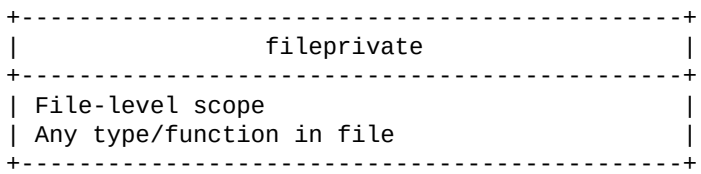
Q: What happens if you mark a type private?
A: If a top-level type is private, it's only usable within the same file and not visible elsewhere.

FILEPRIVATE

Definition:

Visible to any code inside the same .swift file.

Scope Diagram:



Code Example:

```
// File: Room.swift
class Box {
    fileprivate var note = "This file only"
    private var pocket = "secret"
}
class Friend {
    func read() {
        let b = Box()
        print(b.note)   // ■ allowed (same file)
        // print(b.pocket) // ■ not allowed: pocket is private to Box
    }
}
func topLevel() {
    let b = Box()
    print(b.note) // ■ allowed: top-level code in same file can access fileprivate
}
```

Practice Questions (with answers):

Q: What does 'fileprivate' allow?
A: Access from any code within the same .swift file, regardless of type boundaries.

Q: Can code in another file access fileprivate members?
A: No — fileprivate is restricted to the file it is declared in.

Q: Is fileprivate broader than private?
A: Yes — fileprivate is file-scoped while private is type-scoped.

Q: When is fileprivate useful?
A: When multiple helper types or top-level functions in a file need to share implementation details.

Q: Does fileprivate work across modules?
A: No — it's limited to the same .swift file and module boundaries don't change that.

Q: Can extensions in other files access fileprivate?
A: No — only code in the same file can access fileprivate members.

Q: Does fileprivate expose members to tests using @testable?
A: No — @testable allows internal access, not fileprivate across files.

Q: Should you use fileprivate frequently?
A: No — prefer private; use fileprivate only when file-level sharing is required.

Q: Can fileprivate be used on top-level functions?
A: Yes — top-level functions in the same file can access fileprivate members.

Q: What happens if you split types into multiple files?
A: fileprivate access will break; consider refactoring or using internal for module-wide access.

Q: Can multiple types in the same file access fileprivate properties?
A: Yes — any type in the same file can access fileprivate members.

Q: Does fileprivate change ABI or API exposure?
A: No — it's compile-time only and does not affect ABI by itself.

Q: Is fileprivate allowed in protocols?
A: Protocol requirements cannot be fileprivate; they have to be at least internal.

Q: What’s a common use-case for fileprivate?
A: Grouping a type and related helpers in one file that need tight collaboration.

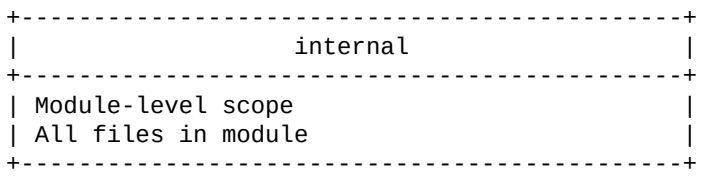
Q: Can a closure in the same file access fileprivate variables?
A: Yes — any closure defined in the same file can access fileprivate members.

INTERNAL

Definition:

Visible anywhere inside the same module/target (default).

Scope Diagram:



Code Example:

```
// Module: MyApp (multiple files)
class Engine {
    var rpm = 1000 // internal by default
    internal func rev() { rpm += 100 }
}
// In another file in same module:
func monitor() {
    let e = Engine()
    print(e.rpm) // ■ allowed in same module
}
```

Practice Questions (with answers):

Q: What is the default access level in Swift?
A: internal is the default when no access modifier is specified.

Q: What does internal allow?
A: Access anywhere within the same module/target across all files.

Q: Can another module access internal members?
A: No — internal is not visible outside its module.

Q: How do tests access internal members?
A: Use @testable import ModuleName in the test target to grant access to internals.

Q: Is internal affected by file location?
A: No — it's module-scoped, so file placement doesn't matter within the module.

Q: When to use internal?
A: For implementation details shared across the app or framework but not part of public API.

Q: Does internal work for Swift packages?
A: Yes — each package product is a module; internal is limited to the package target.

Q: Can you declare an internal protocol?
A: Yes — internal protocols are visible within the module.

Q: Do protocol requirements inherit the protocol's access level?
A: Yes — members cannot be more visible than their declaring type.

Q: If a type is internal and a member is public, what happens?
A: A member cannot be more visible than its type — the effective access is limited by the type.

Q: Does using internal affect library clients?
A: Internals stay hidden from library clients; only public/open types are exposed.

Q: Can internal members be @objc exposed?
A: Yes — you can mark internal members @objc but Objective-C interoperability may require public visibility depending on use.

Q: Is internal suitable for most app code?
A: Yes — use internal by default in app targets.

Q: Can internal be used for stored properties?
A: Yes — stored properties can be internal like any other member.

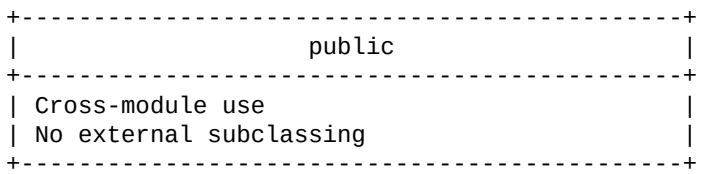
Q: Does internal require explicit keyword?
A: No — it's implicit when no access level is specified.

PUBLIC

Definition:

Visible across modules but cannot be subclassed/overridden outside the module.

Scope Diagram:



Code Example:

```
// In framework MyKit
public class Car {
    public func drive() { print("drive") }      // public method
    public init() {}
}
// In another module (MyApp):
let c = Car() // ■ allowed
c.drive()     // ■ allowed
// class MyCar: Car {} // ■ Error: cannot subclass a public class outside its module
```

Practice Questions (with answers):

Q: What does public allow?
A: Visibility across modules — other modules can use the types and members.

Q: Can public classes be subclassed in other modules?
A: No — public classes cannot be subclassed outside their defining module.

Q: Can public methods be overridden in other modules?
A: No — overriding public methods outside the module is not allowed.

Q: When to use public?
A: Expose API surface of a framework while controlling extensibility.

Q: Does public expose stored properties?
A: Yes — stored properties can be public but consider encapsulation with private(set).

Q: Is public enough to use a type from another module?
A: Yes — public lets you instantiate and call methods across modules.

Q: Do public types appear in generated documentation for frameworks?
A: Yes — public types are part of the framework API surface for clients.

Q: Can a public member be less visible than its type?
A: No — members cannot be more visible than their enclosing type.

Q: Does public affect subclassing within the same module?
A: Within the same module, you can subclass a public class; external modules cannot.

Q: Can an Objective-C client access public Swift APIs?

A: Yes — but Objective-C interoperability may require @objc and specific visibility.

Q: Should library authors prefer public or open?
A: Prefer public for stable APIs; reserve open for explicit extension points.

Q: Can protocols be public?
A: Yes — public protocols are usable by other modules.

Q: Does public allow stored property mutation outside module?
A: Yes, unless you restrict with private(set) or other modifiers.

Q: Is public the default for framework headers?
A: No — default is internal; you must mark as public explicitly for framework exports.

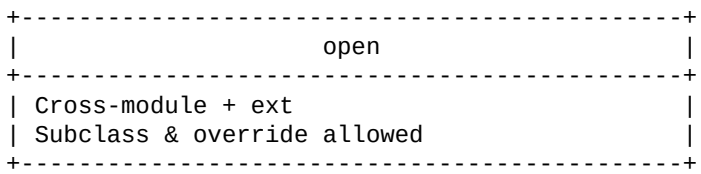
Q: What is a public API breaking change?
A: Changing or removing public types/methods can break clients that depend on the framework.

OPEN

Definition:

Visible across modules and can be subclassed/overridden outside the module.

Scope Diagram:



Code Example:

```
// In framework ShapesKit
open class Shape {
    open func draw() { print("draw") }
    public func info() { print("info") }
}
// In another module:
class MyShape: Shape {                // ■ allowed
    override func draw() { print("custom") } // ■ allowed (draw is open)
}
```

Practice Questions (with answers):

Q: What does open allow?
A: Visibility across modules plus subclassing/overriding outside the module.

Q: Can structs be open?
A: No — open applies only to classes and their members.

Q: If a class is open but method is public, can it be overridden?
A: No — the member must be open to allow overriding outside the module.

Q: Why is open rare?
A: Because it creates extension points that must be carefully designed and maintained.

Q: When should a framework use open?
A: When you explicitly want clients to subclass and customize behavior (e.g., UI components).

Q: Can open classes be subclassed inside the same module?
A: Yes — open allows subclassing both inside and outside the module.

Q: Is open transitive for members?
A: No — members need to be individually declared open to be overridden outside the module.

Q: Can an open class have private members?
A: Yes — open class can still have private/internal members to hide implementation.

Q: Does open affect ABI stability?
A: Not directly — it's an access rule; ABI considerations are separate.

Q: Can you mark initializers open?

A: Initializers cannot be open; they follow visibility rules and subclassing semantics.

Q: Should open methods be designed carefully?
A: Yes — once open, clients can override them; document behavior and constraints.

Q: Does open allow adding stored properties in extensions?
A: No — Swift extensions cannot add stored properties regardless of access level.

Q: Can you make a class open and members public?
A: Yes — but public members won't be overridable outside the module.

Q: Does marking a class open expose it in Obj-C?
A: Objective-C exposure depends on @objc and visibility; open alone doesn't guarantee Obj-C availability.

Q: What is an example of an open system API?
A: UIKit classes like UIView are open to allow app developers to subclass and customize behavior.

Final Summary Table

Specifier	Visibility	Subclass?	Override?	Scope
private	Type-only	No	No	Type
fileprivate	Same file	No	No	File
internal	Same module	Yes (within module)	Yes (within module)	Module
public	Other modules	No	No	Cross-module
open	Other modules	Yes	Yes	Cross-module (extensible)

Appendix: Interview-level Scenarios

Scenario 1: Library API design

You are building a UI framework. Use public for helper utilities that consumers can use but should not extend. Use open for key UI components you intend for consumers to subclass (e.g., base view controllers). Keep internal and private for implementation details. Provide clear documentation and marked extension points.

Scenario 2: Modularising a large app

When splitting into modules, audit internal members: move those needed across modules to public, or refactor to reduce cross-module dependencies. Use facades to expose minimal public APIs.

Scenario 3: Writing unit tests for internal behaviors

Use @testable import ModuleName to access internal members in your test target. Do not change internal to public solely for tests; instead enable testable imports.

Scenario 4: Encapsulation and refactoring

If you find many fileprivate usages in a file, consider whether the file is too large. Prefer private for encapsulation, and extract helper types into their own files or make them internal if module-wide access is needed.

Scenario 5: Designing for extension vs preventing misuse

Mark only the API you intend to be extended as open, and document expected overriding behavior. Use final for classes/methods you want to prevent subclassing or overriding.

Scenario 6: Migrating to Swift Package Manager

Swift packages create modules; ensure access levels are updated—internal stays inside the package product, public/open are required for cross-package usage. Use targets to control visibility.