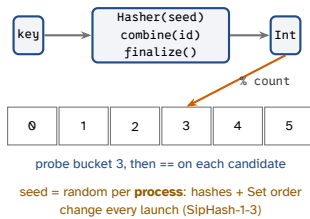


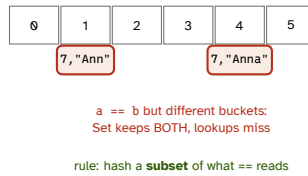
In one line: **Equatable** = value equality (`==`); **Hashable** adds `hash(into:)` under one rule — $a == b \Rightarrow$ **equal hashes** (never the converse); **Comparable** adds a strict total order from **only** `<`. `Set/Dictionary` find a key by *hash first, then ==* — break the contract or mutate a key and elements vanish.

A how a lookup works



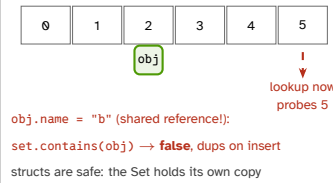
B hash uses MORE than ==

`==` compares id; hash combines id + name



C mutating a class key in a Set

inserted when `obj.name = "a"` → bucket 2



How it works

- **Equatable**: reflexive, symmetric, transitive; `!=` is free. **Synthesized** (SE-0185, Swift 4.1) when you declare it and every stored property / associated value conforms; payload-less enums get it without declaring. Never for **classes**. Must be declared in the type's own file.
- **Hashable**: **Equatable**. Implement `hash(into: inout Hasher)` and combine the same fields `==` reads (SE-0206, Swift 4.2); implementing `hashValue` is deprecated. `Hasher` = SipHash-1-3, randomly seeded per process (HashDoS resistance); `SWIFT_DETERMINISTIC_HASHING=1` fixes the seed for tests. **Never persist** a hash value.
- **Comparable**: **Equatable**. Only `<` is required; `>`, `<=`, `>=`, `min/max`, `sort()`, ranges `a ... b` come free. Must be a strict total order consistent with `==` (`!(a < b) && !(b < a) ⇒ a == b`). Synthesized only for **enums** without raw values whose payloads are Comparable — declaration order (SE-0266, 5.3). Tuples (arity ≤ 6) compare lexicographically: delegate to them.
- **Identifiable** (SE-0261, 5.1): `var id: ID` where ID: Hashable; classes get `ObjectIdentifier(self)` by default. **Identity** (which row is this?) $\neq$ **equality** (did its content change?). SwiftUI keys state, animation and diffing on id.
- **NSObject** bridges: Swift `==` calls `isEqual(_:)`, `hashValue` returns hash. Default `isEqual` is identity. Subclass: override `isEqual(_:)` + `hash` — a custom static `==` or `hash(into:)` is *not* what Set, NSSet or Foundation call.

Example

```
struct User: Hashable { // identity = id
    let id: Int; var name: String
    static func == (l: User, r: User) -> Bool { l.id == r.id }
    func hash(into h: inout Hasher) { h.combine(id) } // same field
}

struct Version: Comparable {
    let major, minor, patch: Int
    static func < (l: Self, r: Self) -> Bool { // only <
        (l.major, l.minor, l.patch) < (r.major, r.minor, r.patch)
    }
}

final class Node: Hashable { // identity
    static func == (l: Node, r: Node) -> Bool { l === r }
    func hash(into h: inout Hasher) {
        h.combine(ObjectIdentifier(self)) // stable
    }
}
```

Which equality for which type?

Type	Choose
value struct	synthesize; hand-write only to exclude fields (cache, timestamps)
entity (User)	<code>==/hash</code> on id; Identifiable for SwiftUI
class, identity	<code>===</code> + <code>ObjectIdentifier</code> (safe to mutate)
class, value-based	only if the hashed fields are <code>let</code>
NSObject subclass	<code>isEqual(_:)</code> + <code>hash</code> , together
normalised keys	<code>==</code> and hash on the SAME normal form (lowercased)

Interview traps

- ! hash combining a field `==` ignores (panel B) — equal values, different buckets.
- ! Mutable class with value-based hash inside a Set/as a dict key (panel C).
- ! `isEqual` overridden, hash forgotten: default hash is identity → broken NSSet/Set.
- ! Set/Dictionary order differs per launch — never assert or persist it.
- ! `Double.nan == .nan` is `false`: breaks reflexivity, `contains(.nan)` fails, `sort` with NaN is unspecified.
- ! `ForEach(items, id: \.self)` = identity is the whole value: editing a row *changes its identity* (state reset, wrong animation); duplicates collide. Array index as id breaks on insert/reorder.
- ! Hash collisions are legal: unequal values may share a hash.

Remember

"Equal $\Rightarrow$ same hash; hash $\subseteq$ equality; never mutate a key; `<` buys all of Comparable; id is who, `==` is what."

Likely questions

1. The Hashable contract? — $a == b \Rightarrow$ equal hashes.
2. Why `hash(into:)` over `hashValue`? — seeded SipHash, no hand mixing.
3. Minimum for Comparable? — `<` (plus `==` from Equatable).
4. Identifiable vs Hashable? — stable identity vs value equality.
5. NSObject in a Set? — override `isEqual(_:)` and `hash`.
6. Mutable class as a dict key? — only with identity hashing or `let` hashed fields.
7. Why does Set order change per launch? — per-process random Hasher seed.