Classes Part 2:
Access, Casts, Any, Generics, and Extensions

Allen Holub

http://holub.com | Allen Holub | @allenholub


RESTRICTED AREA
AUTHORIZED PERSONNEL ONLY

Permissions: file centric, not class centric

`public`	anybody can access
`internal`	anybody in module/ framework/app can access (default)
`private`	accessible from within source file, only


One class per file!
All fields private
All helper methods private
ENIGMA
Klappe
schliessen

MyPlayground

```
public class Super {
    public func doSomething(){}
}

public class Sub : Super {
    override private func doSomething(){}
}
```

```
Playground execution failed: /var/folders/2r/1vl68hpj3sn2qgqbnw_gwcyw0000gn/T/./lldb/2842/playground7.swift:6:27: error: overriding instance method must be as accessible as the declaration
it overrides
    override private func doSomething(){}
             ~~~~~~~      ^
             public
/var/folders/2r/1vl68hpj3sn2qgqbnw_gwcyw0000gn/T/./lldb/2842/playground7.swift:2:17: note: overridden declaration is here
    public func doSomething(){}
                ^
```

MyPlayground

```
public class Super {
    private enum Size { case BIG, SMALL }
    private func doSomething(size: Size){}
}

public class Sub : Super {
    override public func doSomething(size: Size){}
}
```

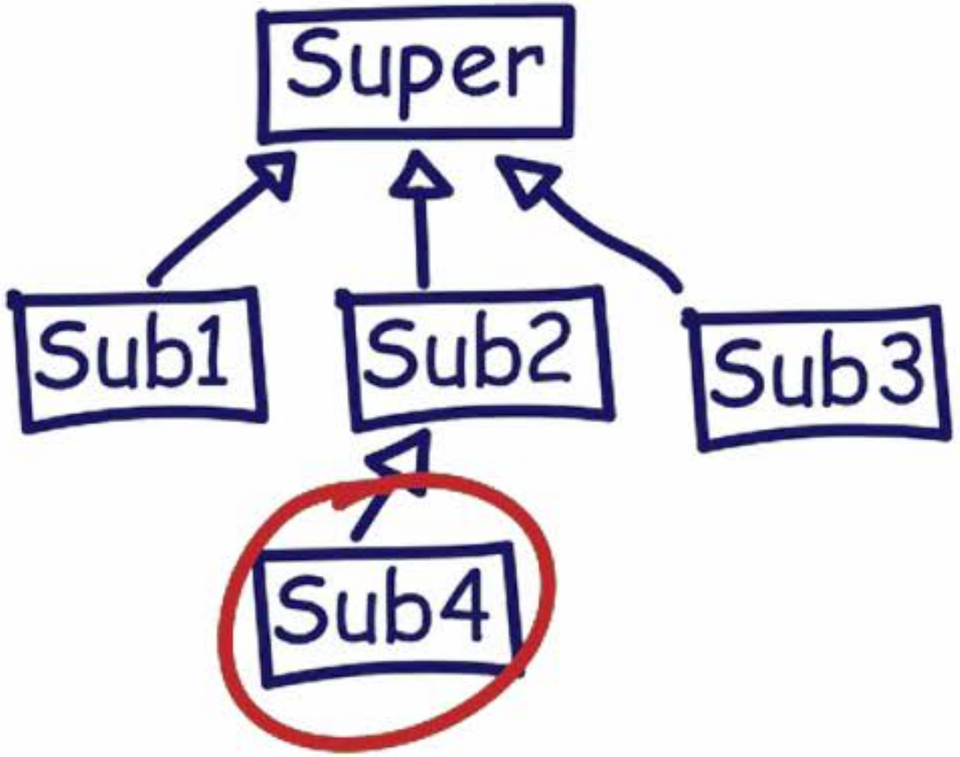
```
Playground execution failed: /var/folders/2r/1vl68hpj3sn2qgqbnw_gwcyw0000gn/T/./lldb/2842/playground12.swift:7:26: error: method cannot be declared public because its parameter uses a
private type
    override public func doSomething(size: Size){}
                         ^                 ~~~~

/var/folders/2r/1vl68hpj3sn2qgqbnw_gwcyw0000gn/T/./lldb/2842/playground12.swift:2:18: note: type declared here
    private enum Size { case BIG, SMALL }
                 ^
```

MyPlayground

```
public class Super {
    public enum Size { case BIG, SMALL }
    private func doSomething(size: Size){}
}

public class Sub : Super {
    override public func doSomething(size: Size){}
}
```

```
public class MyClass {
    public func doSomething(){
        let worker = Helper(owner:self)
        worker.helpMe()
    }
    private func access() {/*...*/}

    private( set )
    public var x: Int {set{ /*...*/} get{return 0} }

    private class Helper {
        private let owner: MyClass
        private init( owner:MyClass ){self.owner = owner}
        private func helpMe() {
            owner.access()
            owner.x = 0
        }
    }
}
```

```
class Super       {/*…*/}
class Sub: Super {/*…*/}

var p :Super = Sub()

if p is Sub {
    var s:Sub = p as! Sub
}


if    let s2 = p as? Sub { /* Use s2 here */ }
```



```
class Super       {/*…*/}
class Sub: Super {/*…*/}

var p :Super = Sub()

if p is Sub {
    var s:Sub = p as! Sub
}


if    let s2 = p as? Sub { /* Use s2 here */ }
guard let s3 = p as? Sub else{ fatalError() }
/* Use s3 here */
```

```swift
import Foundation
var obj:AnyObject = Sub()

let a0:[AnyObject] = ["a", [0,1], Sub()]

for x in a0 {
    if let a = x as? String {print("\(a)")}
    if let b = x as? [Int]  {print("\(b[0])")}
    if let c = x as? Sub    {print("Sub")}
}

let a1:[AnyObject] = ["a", "b", "c"]
for s in a1 as! [String] {
    print(s)
}
```

```
class Customer { var name = "Moe" }
var things: [Any] = [
    0, 42, 0.0, 3.14, "xyz",
    (1,2), Customer(),
    { (x:Int)->Int in return x }
]
```

```
class Customer { var name = "Moe" }
var things: [Any] = [
    0, 42, 0.0, 3.14, "xyz",
    (1,2), Customer(),
    { (x:Int)->Int in return x }
]
```

```
for thing in things {
    switch thing {
    case 0.0          as Double:          print("Double 0.0")
    case              is Double:          print("some double")
    case 0            as Int:             print("Int 0")
    case let i        as Int:             print("\(i)")          // 42
    case let s        as String:          print("\(s)")          // xyz
    case let d as     Double where d>0:    print("\(d)")          // 3.14
    case let (x,y)    as (Int, Int):      print("(\(x),\(y))")  // (1,2)
    case let p        as Customer:        print("\(p.name)")    // Moe
    case let f        as (Int)->Int:      print("\(f(0))")      // abc
    default:                              print("???")
    }
}
```

```swift
func mySwap<T>(inout a:T, inout _ b:T ){
    let tmp = a; a = b; b = tmp
}


var a:Int = 10, b:Int = 20
mySwap( &a, &b )
```

```swift
func mySwap<Int>(inout a:Int , inout _ b: Int ){
    let tmp = a; a = b; b = tmp
}


var a:Int = 10, b:Int = 20
mySwap( &a, &b )
```

```
class Person  {}
class Employee : Person  {
    func daysSinceLastVacation() -> Int {…}
}

class PriorityQueue<ItemT, LevelT> {
  func add( item: ItemT, priorityLevel: LevelT ) {…}
  func getHighestPriorityItem() -> ItemT?         {…}
}

var nextVacation = PriorityQueue<Person,Int>()
func waitForVacation( p: Employee ) {
    nextVacation.add( p,
          priorityLevel: p.daysSinceLastVacation())
}
```

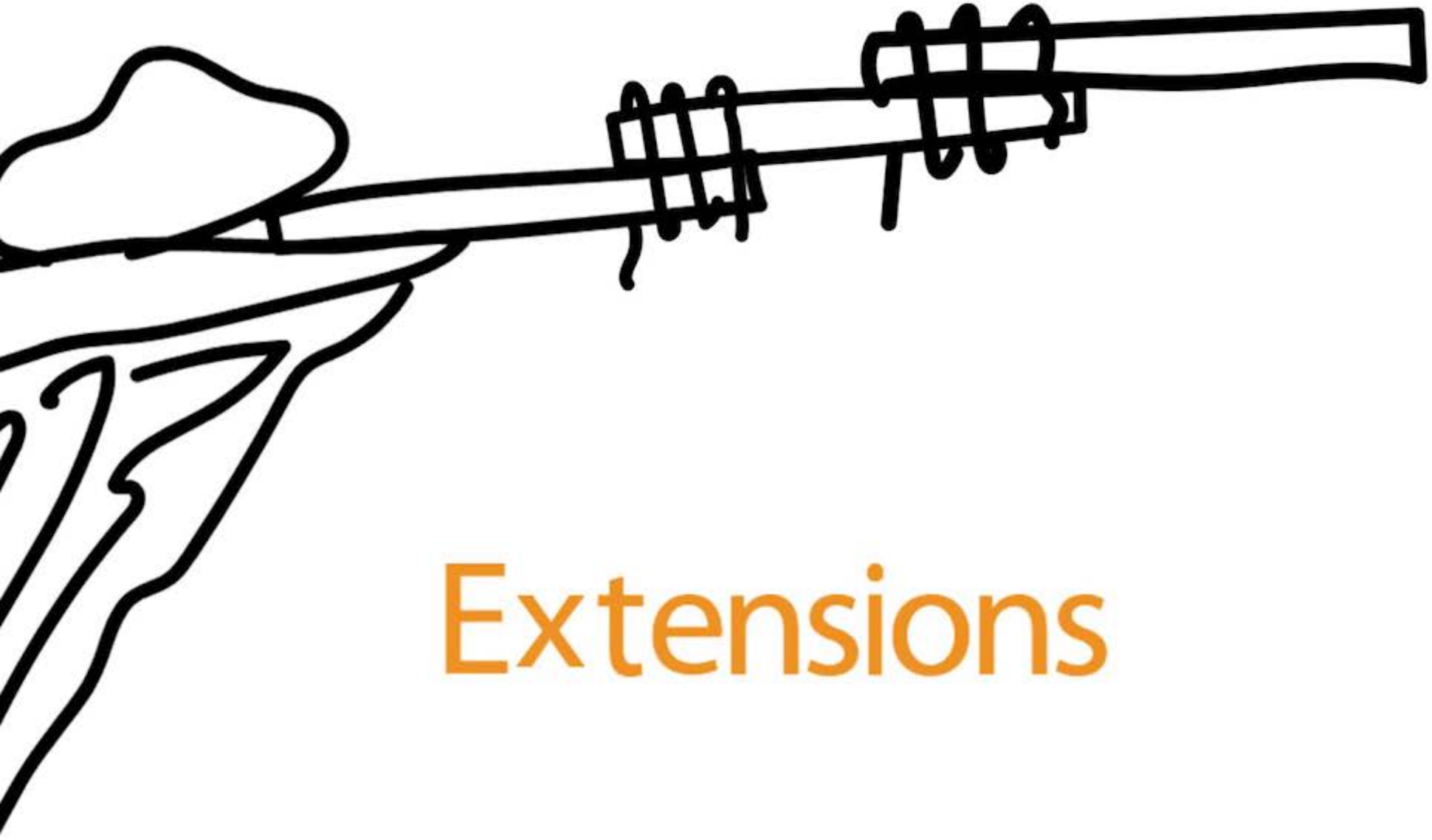
```
enum Bounds <T> {
     case MIN(T)
     case MID(T)
     case MAX(T)
}

let min:  Bounds<Int>

min = .MIN(0)
```

```
enum Bounds <T> {
     case MIN(T)
     case MID(T)
     case MAX(T)
}

let min:  Bounds<Int> = .MIN(0)
```

```
enum Bounds <T> {
     case MIN(T)
     case MID(T)
     case MAX(T)
}

let min:  Bounds = .MIN(0)
```

```
enum Bounds <T> {
     case MIN(T)
     case MID(T)
     case MAX(T)
}

let min = Bounds<Int>.MIN(0)
```

```
enum Bounds <T> {
     case MIN(T)
     case MID(T)
     case MAX(T)
}

let min = Bounds.MIN(0)
```



Extensions

```swift
extension String {
    var length: Int { return self.characters.count }
}


let me = "Allen"
me.length
```

```swift
extension Double {
    var m:  Double { return self }
    var km: Double { return self * 1_000 }
    var mm: Double { return self/1_000.0 }
    var ft: Double { return self/3.2808 }
}

let oneMeter  = 1.0                 // 1.0
let oneInch   = 25.4.mm             // 0.0254
let threeFeet = 3.ft                // 0.91439
let marathon  = 42.km + 195.m       // 42,195.0
```

```
extension Array {
    func elementAt(i: Int, defaultsTo:T) ->T {
        if( 0..<count ~= i ) {
            return self[i]
        }
        return defaultsTo
    }
}

let x = [0, 1, 2]
let y = x.elementAt(3, defaultsTo: 0)
```

```
extension Int {
  func times ( task: ()->() ) {
    for i in 0..<self {
      task()
    }
  }
}

3.times{ print("Hello") }
```

```
class MyClass {
  var contents = ""
  /*…*/

  func isEqual( other: MyClass? ) -> Bool {
    guard let compareTo =   other else { return false } // other is nil
    if          compareTo === self       { return true  } // x.isEqual(x)
    return contents == compareTo.contents
  }
}
```