

TEB



iOS Programlama Eğitimi 3. Gün

Access Control



Erişim Seviyeleri

- public
- internal (default)
- private

Access Control Syntax



```
public class SomePublicClass {}  
internal class SomeInternalClass {}  
private class SomePrivateClass {}  
  
public var somePublicVariable = 0  
internal let someInternalConstant = 0  
private func somePrivateFunction() {}
```

Örnekler



```
public class SomePublicClass {  
    // explicitly public class  
  
    public var somePublicProperty = 0  
    // explicitly public class member  
  
    var someInternalProperty = 0  
    // implicitly internal class member  
  
    private func somePrivateMethod() {}  
    // explicitly private class member  
}
```

Subclassing



- Bir subclass kendi super class'ından daha yüksek bir erişim seviyesine sahip olamaz.
- Örnek: internal superclass'tan public subclass türetilemez

Overriding



- Override edilmiş bir metod superclass'takinden daha erişilebilir olabilir.

```
public class A {  
    private func someMethod() {}  
}
```

```
internal class B: A {  
    override internal func someMethod() {}  
}
```

Getter & Setter



```
class EquilateralTriangle: NamedShape {
    var sideLength: Double = 0.0

    init(sideLength: Double, name: String) {
        self.sideLength = sideLength
        super.init(name: name)
        numberOfSides = 3
    }

    var perimeter: Double {
        get {
            return 3.0 * sideLength
        }
        set {
            sideLength = newValue / 3.0
        }
    }

    override func simpleDescription() -> String {
        return "An equilateral triangle with sides of length \(sideLength)."
    }
}
```

willSet & didSet



```
class StepCounter {  
    var totalSteps: Int = 0 {  
        willSet(newTotalSteps) {  
            println("About to set totalSteps to \$(newTotalSteps)")  
        }  
        didSet {  
            if totalSteps > oldValue {  
                println("Added \$(totalSteps - oldValue) steps")  
            }  
        }  
    }  
}
```


Global & Local Variables



```
struct SomeStructure {  
    static var storedTypeProperty = "Some value."  
    static var computedTypeProperty: Int {  
        // return an Int value here  
    }  
}
```

Global & Local Variables



```
enum SomeEnumeration {  
    static var storedTypeProperty = "Some value."  
    static var computedTypeProperty: Int {  
        // return an Int value here  
    }  
}
```

Global & Local Variables



```
class SomeClass {  
    class var computedTypeProperty: Int {  
        // return an Int value here  
    }  
}
```

Global & Local Variables



```
println(SomeClass.computedTypeProperty)  
// prints "42"
```

```
println(SomeStructure.storedTypeProperty)  
// prints "Some value."
```

```
SomeStructure.storedTypeProperty = "Another value."  
println(SomeStructure.storedTypeProperty)  
// prints "Another value."
```

Audio Channel Sample



Audio Channel Sample



```
struct AudioChannel {  
    static let thresholdLevel = 10  
    static var maxInputLevelForAllChannels = 0  
    var currentLevel: Int = 0 {  
        didSet {  
            if currentLevel > AudioChannel.thresholdLevel {  
                // cap the new audio level to the threshold level  
                currentLevel = AudioChannel.thresholdLevel  
            }  
            if currentLevel > AudioChannel.maxInputLevelForAllChannels {  
                // store this as the new overall maximum input level  
                AudioChannel.maxInputLevelForAllChannels = currentLevel  
            }  
        }  
    }  
}
```

Audio Channel Sample



```
var leftChannel = AudioChannel()  
leftChannel.currentLevel = 7  
println(leftChannel.currentLevel)  
// prints "7"  
println(AudioChannel.maxInputLevelForAllChannels)  
// prints "7"
```

Audio Channel Sample



```
var rightChannel = AudioChannel()  
rightChannel.currentLevel = 11  
println(rightChannel.currentLevel)  
// prints "10"  
println(AudioChannel.maxInputLevelForAllChannels)  
// prints "10"
```


Subscripts



1. `var anArray:Array<Int> = [1,2,4]`
2. `println(anArray[2]) // 4`
3. `anArray[2] = 3`
4. `println(anArray[2]) // 3`

Subscripts



```
subscript(index: Int) -> Int {  
    get {  
        // return an appropriate subscript value here  
    }  
    set(newValue) {  
        // perform a suitable setting action here  
    }  
}
```

Subscripts



```
subscript(index: Int) -> Int {  
    // return an appropriate subscript value here  
}
```

Subscripts



```
struct TimesTable {  
    let multiplier: Int  
    subscript(index: Int) -> Int {  
        return multiplier * index  
    }  
}  
  
let threeTimesTable = TimesTable(multiplier: 3)  
println("six times three is \((threeTimesTable[6]))")  
// prints "six times three is 18"
```

Subscripts



```
struct Matrix {  
    let rows: Int, columns: Int  
    var grid: [Double]  
    init(rows: Int, columns: Int) {  
        self.rows = rows  
        self.columns = columns  
        grid = Array(count: rows * columns, repeatedValue: 0.0)  
    }  
    func isValidForRow(row: Int, column: Int) -> Bool {  
        return row >= 0 && row < rows && column >= 0 && column < columns  
    }  
    subscript(row: Int, column: Int) -> Double {  
        get {  
            assert(isValidForRow(row, column: column), "Index out of range")  
            return grid[(row * columns) + column]  
        }  
        set {  
            assert(isValidForRow(row, column: column), "Index out of range")  
            grid[(row * columns) + column] = newValue  
        }  
    }  
}
```

Subscripts



```
var matrix = Matrix(rows: 2, columns: 2)
matrix[0, 1] = 1.5
matrix[1, 0] = 3.2
```

Subscripts



```
let someValue = matrix[2, 2]  
// this triggers an assert  
// because [2, 2] is outside of the matrix bounds
```