TEB

iOS Programlama Eğitimi
2. Gün

Enumerations

```
enum CompassPoint {
    case North
    case South
    case East
    case West
}
```

Enumerations

1. var direction = CompassPoint.North
2. direction = .West

Enumerations

```
enum Planet {
    case Mercury, Venus, Earth, Mars, Jupiter
    case Saturn, Uranus, Neptune
}
```

Enumerations

```
switch directionToHead {
case .North:
    println("Lots of planets have a north")
case .South:
    println("Watch out for penguins")
case .East:
    println("Where the sun rises")
case .West:
    println("Where the skies are blue")
}
```

Enumerations

```
enum Barcode {
    case UPCA(Int, Int, Int, Int)
    case QRCode(String)
}
```

Enumerations

1. var productBarcode = Barcode.UPCA(8, 8509, 126, 3)
2. productBarcode = .QRCode("ABCDEFGHIJOP")

Enumerations

```
switch productBarcode {
case let .UPCA(numberSystem, manufacturer, product, check):
    println("UPC-A: \(numberSystem), \(manufacturer).")
case .QRCode(let productCode):
    println("QR code: \(productCode).")
}
```

Enum Raw Values

- enum ASCIIControlCharacter: Character {
-     case Tab = "\t"
-     case LineFeed = "\n"
-     case CarriageReturn = "\r"
- }

Enum Raw Values

- enum Planet: Int {
-     case Mercury = 1, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune
- }

Enum Raw Values

- let earthsOrder = Planet.Earth.rawValue
- // earthsOrder = 3

- let possiblePlanet = Planet(rawValue:7)
- // possiblePlanet’in tipi Planet? ve değeri Planet.Uranus

Enum Raw Values

```
let gezegenNo = 9
if let somePlanet = Planet(rawValue:gezegenNo) {
    switch somePlanet {
    case .Earth:
        println(”Dünyamız ☺")
    default:
        println(”Dünya dışında bi gezegen işte")
    }
} else {
    println("\(gezegenNo) nolu bir gezegen yok!")
}
```

Structures

```
struct Paragraph {
    var text : [String]
    var level : Int
}

var myParagraph = Paragraph(text:
["This is the first sentence in my paragraph.",
 "This is the second sentence in my paragraph.”
],  level:1)
```

Struct vs Class

```
class SomeClass {
    var name: String
    init(name: String) {
        self.name = name
    }
}

var aClass = SomeClass(name: "Bob")
var bClass = aClass // aClass and bClass now reference the same instance!
bClass.name = "Sue"

println(aClass.name) // "Sue"
println(bClass.name) // "Sue"
```

Struct vs Class

```
struct SomeStruct {
    var name: String
    init(name: String) {
        self.name = name
    }
}

var aStruct = SomeStruct(name: "Bob")
var bStruct = aStruct // aStruct and bStruct are two structs with the same value!
bStruct.name = "Sue"

println(aStruct.name) // "Bob"
println(bStruct.name) // "Sue"
```

Protocols

```
protocol ExampleProtocol {
    var simpleDescription: String { get }
    mutating func adjust()
}
```

Adopting Protocol

```
class SimpleClass: ExampleProtocol {
    var simpleDescription: String = "A very simple class."
    var anotherProperty: Int = 69105
    func adjust() {
        simpleDescription += "  Now 100% adjusted."
    }
}
```

Using Protocol

1. var a = SimpleClass()
2. a.adjust()
3. let aDescription = a.simpleDescription

Adopting Protocol

```
struct SimpleStructure: ExampleProtocol {
    var simpleDescription: String = "A simple structure"
    mutating func adjust() {
        simpleDescription += " (adjusted)"
    }
}
```

Using Protocol

1. var b = SimpleStructure()
2. b.adjust()
3. let bDescription = b.simpleDescription

Class Only Protocols

```
protocol SomeClassOnlyProtocol: class {
    // class-only protocol definition goes here
}
```

Extensions

```
extension Int: ExampleProtocol {
    var simpleDescription: String {
        return "The number \(self)"
    }
    mutating func adjust() {
        self += 42
    }
}
7.simpleDescription
```

Extensions

```
extension Double {
    var km: Double { return self * 1_000.0 }
    var m: Double { return self }
    var cm: Double { return self / 100.0 }
    var mm: Double { return self / 1_000.0 }
    var ft: Double { return self / 3.28084 }
}
let oneInch = 25.4.mm
println("One inch is \(oneInch) meters")
// prints "One inch is 0.0254 meters"
let threeFeet = 3.ft
println("Three feet is \(threeFeet) meters")
// prints "Three feet is 0.914399970739201 meters
```

In-Out Parameters

```
func swapTwoInts(inout a: Int, inout b: Int) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

In-Out Parameters

1. var someInt = 3
2. var anotherInt = 107
3. swapTwoInts(&someInt, &anotherInt)
4. println("someInt is now \(someInt), and anotherInt is now \(anotherInt)")
5. // prints "someInt is now 107, and anotherInt is now 3

Generics

```
func swapTwoStrings(inout a: String, inout b: String) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

Generics

```
func swapTwoDoubles(inout a: Double, inout b:
Double) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

Generics

- func swapTwoValues<T>(inout a: T, inout b: T) {
-     let temporaryA = a
-     a = b
-     b = temporaryA
- }

Generics

```
func repeat<ItemType>(item: ItemType, times: Int) -> [ItemType] {
    var result = [ItemType]()
    for i in 0..<times {
        result.append(item)
    }
    return result
}
repeat("knock", 4)
```