
열거형

강사 주영민

열거형(enumeration)

- 그룹에 대한 연관된 값을 정의하고 사용가능한 타입
- 다른 언어와 달리 항목 그자체가 고유의 값으로 해당 항목에 값을 매칭 시킬 필요가 없다.(C계열 언어는 Int타입의 값이 매칭됨)
- 원시값(rawValue)이라는 형태로 실제 값(정수, 실수, 문자등)을 부여 할수 있다.
- 열거형의 이니셜라이즈를 정의 할수 있으며, 프로토콜 준수, 연산프로퍼티, 메소드등을 만들수 있습니다.

열거형 문법

```
enum <열거형 이름> {  
    case <열거 항목1>  
    case <열거 항목2>  
    case <열거 항목3>  
}
```

```
enum CompassPoint {  
    case north  
    case south  
    case east  
    case west  
}
```

```
enum Planet {  
    case mercury, venus, earth,  
        mars, jupiter, saturn,  
        uranus, neptune  
}
```

열거형 값 지정

```
var directionToHead = CompassPoint.west
```

```
directionToHead = .north
```

*각 case값만 들어 갈수 있으며, 선언 후 점(.)문법을 통해 쉽게 다른 값을 설정 할수 있다.

Switch문 사용

```
switch directionToHead {  
    case .north:  
        print("Lots of planets have a north")  
    case .south:  
        print("Watch out for penguins")  
    case .east:  
        print("Where the sun rises")  
    case .west:  
        print("Where the skies are blue")  
}
```

*열거형 모든 case가 제공될때 default값은 제공될 필요가 없다.

Associated Values



```
enum Barcode {  
    case upc(Int, Int, Int, Int)  
    case qrCode(String)  
}
```

Associated Values

```
enum Barcode {  
    case upc(Int, Int, Int, Int)  
    case qrCode(String)  
}
```

<연관 열거형 값지정>

```
var productBarcode:Barcode = Barcode.upc(8, 85909, 51226, 3)
```

```
productBarcode = .qrCode("ABCDEFGHJKLMNOP")
```

Associated Values

```
enum Barcode {  
    case upc(Int, Int, Int, Int)  
    case qrCode(String)  
}
```

<연관 열거형 값 불러오기>

```
switch productBarcode {  
    case .upc(let numberSystem, let manufacturer, let product, let check):  
        print("UPC: \(numberSystem), \(manufacturer), \(product), \(check).")  
  
    case .qrCode(let productCode):  
        print("QR code: \(productCode).")  
}
```


Raw Values

```
enum ASCIIControlCharacter: Character {  
    case tab = "\t"  
    case lineFeed = "\n"  
    case carriageReturn = "\r"  
}
```

항목에 지정 Value Set

```
enum Planet: Int{  
    case mercury=1, venus, earth,  
    mars, jupiter, saturn,  
    uranus, neptune  
}
```

시작 Index 연관 Set

```
enum CompassPoint: String {  
    case north, south, east, west  
}
```

항목 이름 Value Set

Raw Values

- .rawValue 프로퍼티를 통해 원시값을 가져올수 있다.

```
let earthsOrder = Planet.earth.rawValue  
// earthsOrder is 3
```

```
let sunsetDirection = CompassPoint.west.rawValue  
// sunsetDirection is "west"
```

Initializing from a Raw Value

- 원시값 열거형에서는 초기화 함수를 통해 instance를 만들수 있다. (rawValue:값 지정으로 인해 생성)
- 초기화를 통해 만든 인스턴스는 옵션널 변수로 만들어 진다.

```
enum Planet: Int{  
    case mercury=1, venus, earth,  
    mars, jupiter, saturn,  
    uranus, neptune  
}
```

```
let possiblePlanet:Planet = Planet(rawValue: 1)!
```

Recursive Enumerations

- 재귀열거형은 다른 인스턴스 열거형이 Associated Values로 사용되는 열거형이다.
- indirect 키워드를 통해 순환 열거형을 명시할수 있으며, 특정 항목만 사용시 case 앞에, 열거형 전체에 사용될 때는 enum 키워드 앞에 붙이면 된다.

```
enum ArithmeticExpression {  
  case number(Int)  
  indirect case addition(ArithmeticExpression,  
                          ArithmeticExpression)  
  indirect case multiplication(ArithmeticExpression,  
                               ArithmeticExpression)  
}
```

```
indirect enum ArithmeticExpression {  
  case number(Int)  
  case addition(ArithmeticExpression, ArithmeticExpression)  
  case multiplication(ArithmeticExpression,  
                      ArithmeticExpression)  
}
```

Recursive Enumerations 예제

$(5 + 4) * 2.$

```
let five = ArithmeticExpression.number(5)
let four = ArithmeticExpression.number(4)

let sum = ArithmeticExpression.addition(five, four)
let product = ArithmeticExpression.multiplication(sum,
ArithmeticExpression.number(2))
```

Recursive Enumerations 예제

```
let five = ArithmeticExpression.number(5)
let four = ArithmeticExpression.number(4)

let sum = ArithmeticExpression.addition(five, four)
let product = ArithmeticExpression.multiplication(sum,
ArithmeticExpression.number(2))

func evaluate(_ expression: ArithmeticExpression) -> Int {
    switch expression {
    case let .number(value):
        return value
    case let .addition(left, right):
        return evaluate(left) + evaluate(right)
    case let .multiplication(left, right):
        return evaluate(left) * evaluate(right)
    }
}

print(evaluate(product))
```

다양한 예제

//기본 연관 값 열거형

```
enum KqueueEvent {  
    case UserEvent(identifier: UInt, fflags: [UInt32], data: Int)  
    case ReadFD(fd: UInt, data: Int)  
    case WriteFD(fd: UInt, data: Int)  
    case VnodeFD(fd: UInt, fflags: [UInt32], data: Int)  
    case ErrorEvent(code: UInt, message: String)  
}
```

//중첩 열거형

```
enum Wearable {  
    enum Weight: Int {  
        case Light = 1  
        case Mid = 4  
        case Heavy = 10  
    }  
    enum Armor: Int {  
        case Light = 2  
        case Strong = 8  
        case Heavy = 20  
    }  
    case Helmet(weight: Weight, armor: Armor)  
    case Breastplate(weight: Weight, armor: Armor)  
    case Shield(weight: Weight, armor: Armor)  
}
```

다양한 예제 - 초기화

```
enum TemperatureUnit {  
  case kelvin, celsius, fahrenheit  
  init?(symbol: Character) {  
    switch symbol {  
    case "K":  
      self = .kelvin  
    case "C":  
      self = .celsius  
    case "F":  
      self = .fahrenheit  
    default:  
      return nil  
    }  
  }  
}
```


다양한 예제 - 함수

```
enum Wearable {  
    enum Weight: Int {  
        case Light = 1  
    }  
    enum Armor: Int {  
        case Light = 2  
    }  
  
    case Helmet(weight: Weight, armor: Armor)  
  
    func attributes() -> (weight: Int, armor: Int) {  
        switch self {  
        case .Helmet(let w, let a):  
            return (weight: w.rawValue * 2,  
                    armor: a.rawValue * 4)  
        }  
    }  
}  
  
let woodenHelmetProps = Wearable.Helmet(weight: .Light,  
                                          armor: .Light).attributes()  
print (woodenHelmetProps)
```

다양한 예제 - 함수

```
enum Device {  
    case iPad, iPhone, AppleTV, AppleWatch  
    func introduced() -> String {  
        switch self {  
            case .AppleTV:  
                return "\(self) was introduced 2006"  
            case .iPhone:  
                return "\(self) was introduced 2007"  
            case .iPad:  
                return "\(self) was introduced 2010"  
            case .AppleWatch:  
                return "\(self) was introduced 2014"  
        }  
    }  
}  
  
print (Device.iPhone.introduced())
```

다양한 예제 - 연산프로퍼티

```
enum Device {  
    case iPad, iPhone  
    var year: Int {  
        switch self {  
            case .iPhone:  
                return 2007  
            case .iPad:  
                return 2010  
        }  
    }  
}  
  
print (Device.iPhone.year)
```

옵셔널

nil이란?

- 아무것도 없는 상태
- 변수만 선언되어 있으며, 아직 instance가 할당되기 전

Type Safety

- nil인 상태에서 속성을 참조하거나, 함수를 실행시 발생하는 error로 인한 코드의 불안정성 내포
- Swift의 중요한 특징 중 하나는 Safety!!
- Type Safety를 위해 컴파일러 수준의 nil 체크
- 만약 nil인 변수 선언을 해야할 경우 optional을 사용한다.
- optional은 두가지 가능성을 가질수 있는데
한개는 값이 있음(nil이 아님을 확신)을 나타내고 (!기호 사용)
또다른 한가지는 nil일 가능성을 내포하고 있다.(?기호 사용)

옵셔널 타입

`var num: Int`



프로퍼티를 초기화없이
선언시 컴파일에러

?

`var num: Int?`
(Optional Int)

옵셔널 타입

```
public enum Optional<Wrapped> : ExpressibleByNilLiteral {  
    case none  
    case some(Wrapped)  
    public init(_ some: Wrapped)  
  
}
```

```
let shortForm: Int? = Int("42")  
let longForm: Optional<Int> = Int("42")
```

```
var num: Int? = Optional.none  
var num: Int? = Optional.some(3)
```


Unwrapping

Optional 변수에 값이 있음을 확인하여 일반 변수로 전환해준다.

- Forced Unwrapping
- Optional Binding
- Early Exit

강제 해제(Forced Unwrapping)

```
func testFuc(optionalStr:String?)  
{  
    if optionalStr != nil  
    {  
        let unwrapStr:String = optionalStr!  
        print(unwrapStr)  
    }  
}
```

Optional Binding

```
func testFuc(optionalStr:String?)  
{  
    if let unwrapStr = optionalStr  
    {  
        print(unwrapStr)  
    }  
}
```

Optional Binding

*문제 : Optional 바인딩 할 갯수가 한개가 아니라면?

```
func isNumber(inputNum1:Int?, inputNum2:Int?) -> Bool
{
}
}
```

Optional Binding

```
func isNumber(inputNum1:Int?, inputNum2:Int?) -> Bool
{
    if let firstNumber = inputNum1,
       let secondNumber = inputNum2
    {
        return true
    } else
    {
        return false
    }
}
```

* (,) 콤마를 통해 옵셔널 바인딩을 추가 할수 있다.

Early Exit

```
guard 조건값 else  
{  
    //조건값이 거짓일때 실행  
}
```

Early Exit

```
func testFuc(optionalStr:String?)  
{  
    guard let unwrapStr:String = optionalStr else  
    {  
        return  
    }  
    print(unwrapStr)  
}
```

Early Exit - 예제

```
func getFriendList(list:[String]?)
{
    guard let list = list else { return }

    for name in list
    {
        if name == "joo" {
            print("find")
        }
    }
}
```


Optional Chaining

- 인스턴스의 프로퍼티나 메소드에 접근하기 위해 옵셔널 체인 연산자를 통해 접근합니다.

```
var displayLabel: UILabel?  
displayLabel?.text = "displayLabel에 옵셔널 체이닝 사용"
```

nil-coalescing

- nil값일 경우 디폴트 값을 지정해 줄수 있다.
- ?? 기호를 사용

```
let defaultImagePath = "/images/default.png"  
let heartPath = imagePaths["image"] ?? defaultImagePath  
  
let shapePath = imagePaths["image"] ?? imagePaths["sub"] ??  
defaultImagePath
```