열거형

강사 주영민

Fast campus

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.upc(8, 85909, 51226, 3)

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

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).")
}
```

Associated Values

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

<Pattern Matching>

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

if case let Barcode.upc(8, companyCode, productCode, 3) =
productBarcode
{
    //정상 바코드
    print(companyCode)
    print(productCode)
}
```

Raw Values

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

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

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

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)
```

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 - 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)
```

에러처리

강사 주영민

예외처리

- 프로그램의 오류 조건에 응답하고 오류 조건에서 복구하는 프로세스입니다

- Swift는 런타임시 복구 가능한 오류를 던지고, 포착하고, 전파하고, 조작하는 기능을 제공합니다.

- 에러는 Error 프로토콜을 준수하는 유형의 값으로 나타냅니다. 실제로 Error프로토콜은 비어 있으나 오류를 처리할수 있는 타입임을 나타냅니다.

열거형의 에러 표현

- 열거형은 에러를 표현하는데 적합합니다.

```
enum VendingMachineError: Error {
    case invalidSelection
    case insufficientFunds(coinsNeeded: Int)
    case outOfStock
}
```

에러발생

- throw 키워드를 통해 에러를 발생 시킵니다.

```
throw VendingMachineError.insufficientFunds(coinsNeeded: 5)
```

에러전달

- 함수의 작성중 에러가 발생할수 있는 함수에는 매개변수 뒤에 throws 키워드를 작성하여 에러가 전달될수 있는 함수를 선언합니다.

```
//에러전달 가능성 함수
func canThrowErrors() throws -> String

//에러전달 가능성이 없는 함수
func cannotThrowErrors() -> String
```

에러처리

- 함수가 에러를 throw하면 프로그램의 흐름이 변경되므로 에러가 발생할 수있는 코드의 위치를 신속하게 식별 할 수 있어야합니다.

- 이 장소를 식별하기 위해 try 나 try?, try!를 사용할수 있습니다.

- 발결된 에러를 처리하기 위해 do-catch 문을 사용해서 에러를 처리 합니다.

do - catch

```
do {
    try expression
    statements
} catch pattern 1 {
    statements
} catch pattern 2 where condition {
    statements
}
```

Converting Errors to Optional Value

```swift
func someThrowingFunction() throws -> Int {
    // ...
}

let x = try? someThrowingFunction()
```

… is …

```swift
let y: Int?
do {
    y = try someThrowingFunction()
} catch {
    y = nil
}
```

Data 저장

강사 주영민

1. file - PList(XML형식)
2. file 형태의 dB -sqlite
3. userDefault
4. PList
3. DB : 여러개의 파일
4. network : DB인데 앱 외부에 있는 것을 갖고 옴

Property list

Key	Type	Value
▼ Information Property List	Dictionary	(14 items)
Localization native development re...	String	en
Executable file	String	$(EXECUTABLE_NAME)
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	????
Bundle version	String	1
Application requires iPhone enviro...	Boolean	YES
Launch screen interface file base...	String	LaunchScreen
Main storyboard file base name	String	Main
▼ Required device capabilities	Array	(1 item)
Item 0	String	armv7
▶ Supported interface orientati...	Array	(3 items)

Property list - plist

- 심플한 “파일” 저장 방법 중 하나.
- Key,Value구조로 데이터 저장
- File 형태로 저장되다 보니 외부에서 Access가능(보안 취약)

파일 위치

- 파일이 저장되는곳 Bundle & Documents 폴더
  번들: 리소스가 저장되는 장소, 인스톨 될때 같이 설치　　Document: 사용되는 순간 말들어짐, 사용자마다 다름
- Bundle은 프로젝트에 추가된 Resorce가 모인 곳
- 프로그램이 실행되며 저장하는 파일은 Documents폴더에 저장된다.
- **즉! plist파일의 데이터만 불러오는 역할은 Bundle을 통해서, plist파일에 데이터를 쓰고 불러오는 역할은 Documents폴더에 저장된 파일로!**

Plist File In Bundle

1. bundle에 있는 파일 Path 가져오기
2. Path를 통해 객체로 변환, 데이터 불러오기
3. 사용

Bundle

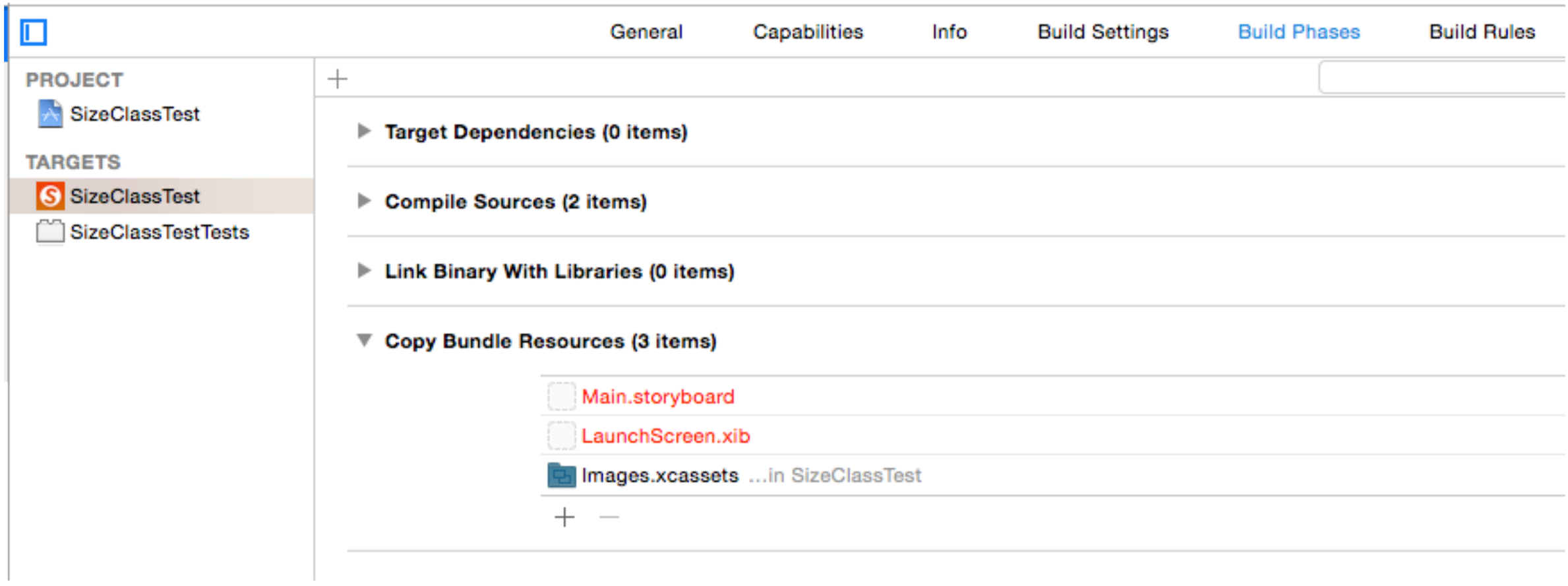
강사 주영민

Fast campus

Bundle

- 실행코드, 이미지, 사운드, nib 파일, 프레임 워크,설정파일 등 코드와 리소스가 모여있는 file system내의 Directory

Bundle 리소스 확인



Main Bundle 가져오기

```swift
// Get the main bundle for the app.
let mainBundle = Bundle.main
```

리소스 파일 주소 가져오기

```
// Get the main bundle for the app.
let mainBundle = Bundle.main
let filePath:String? = mainBundle.path(forResource:
"fileName", ofType: "rType")
```

데이터 불러오기

```
// Get the main bundle for the app.
let mainBundle = Bundle.main
let filePath:String? = mainBundle.path(forResource:
"fileName", ofType: "rType")

if let path = filePath {
    let image = UIImage(contentsOfFile: path)
}
```

Plist File load for Bundle

```
func loadPlist(fileName:String)
{
    //Path
    let bundlePath = Bundle.main.path(forResource: fileName, ofType: "plist")

    //Data
    if let data = try? Data(contentsOf: URL(fileURLWithPath: bundlePath!))
    {
       //Dictionary
        let dic = try? PropertyListSerialization.propertyList(from: data,
                                                              options: .mutableContainersAndLeaves, format: nil)
        print(dic)
    }
}
```

Plist파일 불러오기

- 앱내 저장되어 있는 plist 리소스 파일의 데이터는 번들을 통해 가져올 수 있다.

Plist File In Document

1. Document folder Path 찾기
2. Document folder에 plist 파일이 있는지 확인
    - 만약 없다면 : bundle에 있는 파일 Document에 복사
3. Path를 통해 Data인스턴스로 변환
4. PropertyListSerialization으로 Data를 컬렉션으로 변환
5. 컬렉션 데이터 가공 후 PropertyListSerialization로 Data 변환
6. 파일에 다시 저장

1. 파일 불러오기 (NSSearchPathForDirectoriesInDomains)

```
let rootPath:String = NSSearchPathForDirectoriesInDomains(.documentDirectory,
.userDomainMask, true)[0]
let fullPath:String = rootPath + "/" + fileName + ".plist"
```

- doucument 폴더에 Path구하기

2. Document folder에 파일 있는지 확인

```
if !FileManager.default.fileExists(atPath: fullPath)
{

}
```

- doucument 폴더에 plist파일이 존재 하지는지 확인

3. bundle에 있는 파일 Document에 복사

```
if !FileManager.default.fileExists(atPath: fullPath)
{
    if let bundlePath:String = Bundle.main.path(forResource: fileName, ofType:
    “plist")
    {
         try? FileManager.default.copyItem(atPath: bundlePath, toPath: fullPath)
    }
}
```

- 만약 doucument에 해당 plist파일이 존재 하지 않을때, bundle에 있는 파일을 document폴더로 복사

4. Dictionary 인스턴스 불러오기

```
if let data = try? Data(contentsOf: URL(fileURLWithPath: fullPath))
{
    do
    {
        if var dic:[String:Any] = try PropertyListSerialization.propertyList(from: data,
                    options: .mutableContainersAndLeaves, format: nil) as? [String:Any]
        {



        }
    }catch
    {
        print("error")
    }
}
```

5. write(to)메소드를 통해 파일에 저장

```
if var dic:[String:Any] = try PropertyListSerialization.propertyList(from: data,
                    options: .mutableContainersAndLeaves, format: nil) as? [String:Any]
{
    dic.updateValue(3, forKey: "hoho")

    let newData = try PropertyListSerialization.data(fromPropertyList: dic,
                                                     format: .xml, options: 0)
    try newData.write(to: URL(fileURLWithPath: fullPath))

}
```

실습

- 한번 같이 만들어 볼까요?