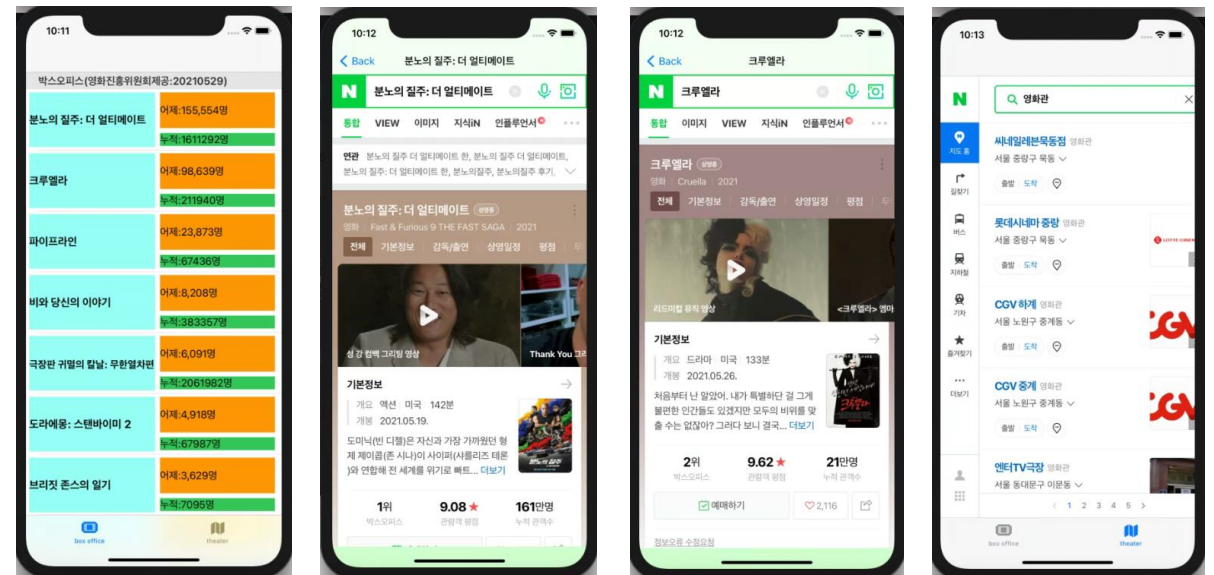


프로토콜 (protocol)과 Delegate

신입 과제

- 공공DB(Open API)를 Restful하게 가져와서 JSON parsing해서 테이블 뷰에 보여주세요.
- 실무 iOS 앱 개발 시 가장 많이 사용하고 중요한 기술
 - 테이블 뷰, 네트워크 통신, 비동기 처리, 웹뷰, 탭바, 맵킷, autolayout
 - iOS 프레임워크 및 디자인 패턴
 - Array, Dictionary 등 활용
- 네트워크 통신(SOAP vs. RESTful)
- JSON parsing(XML vs. JSON)
- Open API 활용한 앱 개발
 - Open API 사용 신청 및 인증 키 발급
 - 공공DB
 - <https://www.data.go.kr/>
 - SK 플레닛, Daum, Naver, Google 등



delegate

- 대표자(명사)
- 위임하다(동사)

명사

1. (집단의 의사를 대표하는) 대표(자)

The conference was attended by **delegates** from 56 countries.



그 회의에는 56개국의 대표들이 참석했다.

동사

1. (권한업무 등을) 위임하다

[V] Some managers find it difficult to **delegate**.



일부 관리자들은 (권한을) 위임하는 것을 힘들어 한다.

문형 ~ (sth) (to sb)

2. 타동사 [VN to inf 주로 수동태로]
(대표를) 뽑다[선정하다]

I've been **delegated** to organize the Christmas party.



내가 크리스마스 파티를 준비하도록 뽑혔다.

Delegation

- <https://docs.swift.org/swift-book/LanguageGuide/Protocols.html>
- Delegation is a **design pattern** that enables a class or structure to hand off (or delegate) some of its responsibilities to an instance of another type.
- This design pattern is implemented by defining a protocol that encapsulates the delegated responsibilities, such that a conforming type (known as a delegate) is guaranteed to provide the functionality that has been delegated.
- Delegation can be used to respond to a particular action, or to retrieve data from an external source without needing to know the underlying type of that source.
- Delegation(위임)은 클래스나 구조체가 **일부 책임을 다른 유형의 인스턴스로 전달 (또는 위임)**할 수 있도록 하는 디자인 패턴이다.
- 위임된 기능은 프로토콜에서 정의하며, **delegate가 위임된 기능을 제공한다.**
- 위임은 특정 작업에 응답하거나 외부에서 데이터를 가져오는데 사용할 수 있다.

delegate

- 대리자, 조력자
- 이런 일이 있을 때 delegate 너가 좀 전담해줘
 - 델리게이트로 선언된 (보통 내가 만든 클래스의) 객체는 자신을 임명한 객체(테이블뷰, 피커뷰 등)가 일을 도와달라고 하면 지정된 메서드를 통하여 처리해 줌
- 델리게이트 패턴
 - 하나의 객체가 모든 일을 처리하는 것이 아니라 처리 해야 할 일 중 일부를 다른 객체에 넘기는 것
- 보통 프로토콜을 사용

프로토콜(protocol)

■ 특정 클래스와 관련없는 프로퍼티, 메서드 선언 집합

- 함수(메서드) 정의는 없음
- 기능이나 속성에 대한 설계도
- 클래스(구조체, 열거형)에서 채택(adopt)하여 메서드를 구현해야 함

■ 자바, C#의 interface

■ C++의 abstract base class

■ Protocol Oriented Programming(POP)

- 프로토콜 단위로 묶어 표현하고, extension으로 기본적인 것을 구현(protocol default implementation)을 해서 단일 상속의 한계를 극복

스위프트 상속과 프로토콜 채택

■class 자식:부모 { }

- 부모 클래스는 하나만 가능하며 여러 개라면 나머지는 프로토콜

■ class 클래스명:부모명, 프로토콜명{}

- 부모가 있으면 부모 다음에 표기

■ class 클래스명:부모명, 프로토콜명1,프로토콜명2 {}

■ class 클래스명:프로토콜명{}

- 부모가 없으면 바로 표기 가능

```
class ViewController: UIViewController, UIPickerViewDelegate,  
                    UIPickerViewDataSource {
```

■ class 클래스명:프로토콜명1, 프로토콜명2{}

■ 클래스, 구조체, 열거형, extension에 프로토콜을 채택(adopt)할 수 있다.

- 상속은 클래스만 가능

protocol 정의

```
protocol 프로토콜명{
```

```
    프로퍼티명
```

```
    메서드 선언 //메서드는 선언만 있음
```

```
}
```

```
protocol 프로토콜명 : 다른프로토콜, 다른프로토콜2{
```

```
    // 프로토콜은 다중 상속도 가능
```

```
}
```


프로토콜과 프로퍼티/메서드 선언

```
protocol SomeProtocol {  
    var x: Int { get set } //읽기와 쓰기가 가능  
    var y: Int { get } //읽기 전용  
    static var tx: Int { get set }  
    static func typeMethod()  
    func random() -> Double  
}
```

protocol 정의, 채택, 준수

```
protocol Runnable {    //대리하고 싶은 함수 목록 작성
    var x : Int {get set} //읽기와 쓰기 가능 프로퍼티,{get}은 읽기 전용
    //property in protocol must have explicit { get } or { get set } specifier
    func run()           //메서드는 선언만 있음
}

class Man : Runnable {    //채택, adopt
    var x : Int = 1        //준수, conform
    func run(){print("달린다~")} //준수, conform
}
```

■ class Man에 x, run()정의 없다면

- type 'Man' does not conform to protocol 'Runnable'

상속과 프로토콜 채택(adopt)

- ViewController클래스는 부모 UIViewController를 상속받고, UIPickerView형의 인스턴스 pickerView를 선언

```
class ViewController: UIViewController{  
    @IBOutlet var pickerView : UIPickerView!
```

- 피커뷰 인스턴스를 사용하기 위해 프로토콜 UIPickerViewDelegate와 UIPickerViewDataSource를 채택

```
class ViewController: UIViewController,  
    UIPickerViewDelegate, UIPickerViewDataSource {
```

- 프로토콜 UIPickerViewDelegate와 UIPickerViewDataSource의 필수 메서드는 모두 구현해야 프로토콜을 준수(conform)

Mountain View
Sunnyvale
Cupertino
Santa Clara
San Jose

프로토콜 채택(adopt)하고 위임

```
class ViewController: UIViewController, UIPickerViewDelegate, UIPickerViewDataSource{
```

- <https://developer.apple.com/documentation/uikit/uipickerviewdelegate>
- <https://developer.apple.com/documentation/uikit/uipickerviewdatasource>

Documentation > ... > UIPickerViewDelegate > pickerView(_:rowHeightF...

Instance Method

XXDelegate는 어떤 행동에
대한 반응 동작 메서드

pickerView(_:rowHeightForComponent:)

Called by the picker view when it needs the row height to use for drawing row content.

Declaration

클래스 내에 이 메서드는
필요하면 구현함

```
optional func pickerView(_ pickerView: UIPickerView,  
rowHeightForComponent component: Int) -> CGFloat
```

Documentation > ... > UIPickerViewDataSource > numberOfComponents(i...

Instance Method

XXDataSource에는 데이터를
받아 뷰를 그려주는 메서드

numberOfComponents(in:)

Called by the picker view when it needs the number of components.

Required.

클래스 내에 이 메서드는
반드시 구현해야 함

Declaration

```
func numberOfComponents(in pickerView: UIPickerView) -> Int
```

Mountain View
Sunnyvale
Cupertino
Santa Clara
San Jose

UIPickerViewDelegate

- <https://developer.apple.com/documentation/uikit/uipickerviewdelegate>
- UIPickerView에 대한 프로토콜
- 나는 protocol이야. 널 도와줄 조력자야
- 피커뷰야, 너 혼자 많은 일하기 바쁘지?
- 내 안에 피커뷰 너에게 어떤 일이 일어났을 때 하고 싶은 일들이 메서드 목록(선언)이 있어
- 피커뷰를 사용하는 클래스에서는 우선 나 (피커뷰 델리게이트)를 채택해
- 채택한 클래스에서는 자신이 델리게이트라 지정하는 것 잊지 말고
 - pickerImage.delegate = self
- 구현한 기능은 iOS프레임워크 내부적으로 원하는 시점에 피커뷰가 호출(callback)해
- 그 시점에 하고 싶은 일만 메서드 내부에 구현해

```
public protocol UIPickerViewDelegate : NSObjectProtocol {  
  
    // returns width of column and height of row for each component.  
    @available(iOS 2.0, *)  
    optional public func pickerView(_ pickerView: UIPickerView, widthForComponent component:  
        Int) -> CGFloat  
  
    @available(iOS 2.0, *)  
    optional public func pickerView(_ pickerView: UIPickerView, heightForComponent  
        component: Int) -> CGFloat  
  
    // these methods return either a plain NSString, a NSAttributedString, or a view (e.g  
    // UILabel) to display the row for the component.  
    // for the view versions, we cache any hidden and thus unused views and pass them back for  
    // reuse.  
    // If you return back a different object, the old one will be released. the view will be  
    // centered in the row rect  
    @available(iOS 2.0, *)  
    optional public func pickerView(_ pickerView: UIPickerView, titleForRow row: Int,  
        forComponent component: Int) -> String?  
  
    @available(iOS 6.0, *)  
    optional public func pickerView(_ pickerView: UIPickerView, attributedTitleForRow row:  
        Int, forComponent component: Int) -> NSAttributedString? // attributed title is  
        favored if both methods are implemented
```

```
class ViewController: UIViewController,  
    UIPickerViewDelegate, UIPickerViewDataSource{  
    @IBOutlet var pickerImage : UIPickerView!  
    :  
    pickerImage.delegate = self
```

Mountain View
Sunnyvale
Cupertino
Santa Clara
San Jose

UIPickerViewDelegate 프로토콜의 메서드

```
public protocol UIPickerViewDelegate : NSObjectProtocol {

    // returns width of column and height of row for each component.
    @available(iOS 2.0, *)
    optional public func pickerView(_ pickerView: UIPickerView, widthForComponent component:
        Int) -> CGFloat

    @available(iOS 2.0, *)
    optional public func pickerView(_ pickerView: UIPickerView, rowHeightForComponent
        component: Int) -> CGFloat

    // these methods return either a plain NSString, a NSAttributedString, or a view (e.g
    UILabel) to display the row for the component.
    // for the view versions, we cache any hidden and thus unused views and pass them back for
    reuse.
    // If you return back a different object, the old one will be released. the view will be
    centered in the row rect
    @available(iOS 2.0, *)
    optional public func pickerView(_ pickerView: UIPickerView, titleForRow row: Int,
        forComponent component: Int) -> String?

    @available(iOS 6.0, *)
    optional public func pickerView(_ pickerView: UIPickerView, attributedTitleForRow row:
        Int, forComponent component: Int) -> NSAttributedString? // attributed title is
        favored if both methods are implemented
```

Mountain View
Sunnyvale
Cupertino
Santa Clara
San Jose

Setting the Dimensions of the Picker View

```
func pickerView(UIPickerView, rowHeightForComponent: Int) ->
CGFloat
    Called by the picker view when it needs the row height to use for drawing row content.

func pickerView(UIPickerView, widthForComponent: Int) -> CGFloat
    Called by the picker view when it needs the row width to use for drawing row content.
```

컴포넌트의 높이가 궁금할 때 피커뷰가 호출

Setting the Content of Component Rows

The methods in this group are marked @optional. However, to use a picker view, you must implement either the `pickerView(_:_:titleForRow:forComponent:)` or the `pickerView(_:_:viewForRow:forComponent:reusing:)` method to provide the content of component rows.

열의 내용이 알고 싶을 때 피커뷰가 호출함

```
func pickerView(UIPickerView, titleForRow: Int, forComponent:
Int) -> String?
    Called by the picker view when it needs the title to use for a given row in a given
    component.
```

```
func pickerView(UIPickerView, attributedTitleForRow: Int, for
Component: Int) -> NSAttributedString?
    Called by the picker view when it needs the styled title to use for a given row in a given
    component.
```

각 열의 view가 알고 싶을 때 피커뷰가 호출하는 메서드

```
func pickerView(UIPickerView, viewForRow: Int, forComponent: Int,
reusing: UIView?) -> UIView
    Called by the picker view when it needs the view to use for a given row in a given
    component.
```

사용자가 룰렛을 선택할 때 피커뷰가 호출하는 메서드

Responding to Row Selection

```
func pickerView(UIPickerView, didSelectRow: Int, inComponent:
Int)
    Called by the picker view when the user selects a row in a component.
```

UIPickerViewDataSource의 필수 메서드는 반드시 구현해야 함

```
public protocol UIPickerViewDataSource : NSObjectProtocol {  
  
    // returns the number of 'columns' to display.  
    @available(iOS 2.0, *)  
    public func numberOfComponents(in pickerView: UIPickerView) -> Int  
  
    // returns the # of rows in each component..  
    @available(iOS 2.0, *)  
    public func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int)  
        -> Int  
}
```

```
func numberOfComponents(in: UIPickerView) -> Int  
    Called by the picker view when it needs the number of components.  
    Required.  
  
func pickerView(UIPickerView, numberOfRowsInComponent: Int) ->  
Int  
    Called by the picker view when it needs the number of rows for a specified component.  
    Required.
```

XXDataSource는 데이터를 받아 뷰를 그려주는 역할

Mountain View
Sunnyvale
Cupertino
Santa Clara
San Jose

inch	
centimeter	
foot	inch
meter	centimeter
mile	foot
	meter

```
func numberOfComponents(in pickerView: UIPickerView) -> Int {  
    return 1  
}  
func pickerView(_ pickerView: UIPickerView,  
    numberOfRowsInComponent component: Int) -> Int {  
    return Array.count  
}
```

<https://developer.apple.com/documentation/uikit/uipickerviewdatasource>

피커뷰 관련 delegate, datasource 그림 그리기

Mountain View
Sunnyvale
Cupertino
Santa Clara
San Jose

```
class ViewController: UIViewController,
UIPickerViewDelegate, UIPickerViewDataSource{
@IBOutlet var pickerImage : UIPickerView!
:
pickerImage.delegate = self
```

<https://developer.apple.com/documentation/uikit/uipickerview/1614379-delegate>

피커뷰 사용하는 클래스(ViewController)

```
class ViewController: UIViewController,
UIPickerViewDelegate, UIPickerViewDataSource{
@IBOutlet var pickerImage : UIPickerView!
:
pickerImage.delegate = self
//내 클래스가 피커뷰의 델리게이트
pickerView.dataSource = self
:
func pickerView(_ pickerView: UIPickerView,
didSelectRow row: Int, inComponent component:
Int){ } //준수, 피커뷰가 필요할 때 호출함
//사용자가 룰렛을 선택할 때마다 피커뷰가 호출
func numberOfComponents(in pickerView:
UIPickerView) -> Int {
return 1
} //준수, 필수 메서드
//피커뷰가 컴포넌트의 수 알고 싶을 때 호출함
```

1

채택

피커뷰 델리게이트 프로토콜

UIPickerViewDelegate protocol

어떤 행동에 대한 반응 동작 메서드 목록

```
optional func pickerView(_ pickerView: UIPickerView, didSelectRow
row: Int, inComponent component: Int)
//채택하더라도 반드시 구현할 필요는 없음
```

채택

피커뷰 데이터소스 프로토콜

UIPickerViewDataSource protocol

데이터를 받아 피커뷰를 그려주는 메서드 목록

```
func numberOfComponents(in pickerView: UIPickerView) -> Int
//채택하면 반드시 구현해야 함
```


extension으로 소스를 깔끔하게

- 아래처럼 소스를 작성하면 클래스가 너무 비대해짐

```
class ViewController: UIViewController, UIPickerViewDelegate,  
UIPickerViewDataSource{}
```

- 소스를 다음과 같이 나누면 깔끔하고 이해하기 쉬움

```
class ViewController: UIViewController{  
extension ViewController: UIPickerViewDelegate{  
extension ViewController: UIPickerViewDataSource{}
```

TableView의 DataSource : UITableViewDataSource프로토콜

- <https://developer.apple.com/documentation/uikit/uitableviewdatasource>

- 필수 메서드 2개

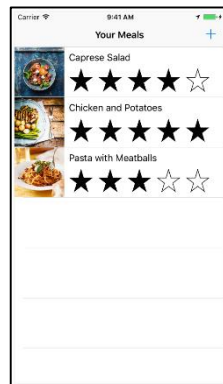
```
// Return the number of rows for the table.
```

```
override func tableView(_ tableView: UITableView, numberOfRowsInSectionSection section: Int) -> Int {  
    return items.count  
}
```

```
// Provide a cell object for each row.
```

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  
    // Fetch a cell of the appropriate type.  
    let cell = tableView.dequeueReusableCell(withIdentifier: "cellTypeIdentifier", for: indexPath)  
    // Configure the cell's contents.  
    cell.textLabel!.text = "Cell text"  
    return cell  
}
```

**과제 : 2개
메서드가 하는 일**

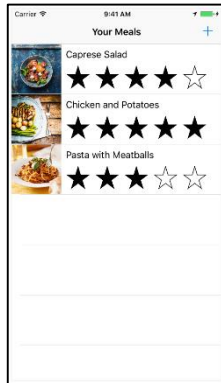


```
override func numberOfSections(in tableView: UITableView) -> Int {  
    // #warning Incomplete implementation, return the number of sections  
    return 1  
} //If you do not implement this method, the table configures the table with one section.
```

TableView의 Delegate: UITableViewDelegate프로토콜

- <https://developer.apple.com/documentation/uikit/uitableviewdelegate>
- header, footer view를 만들고 관리
- 행, header, footer 높이 지정
- 스무스한 스크롤링을 위해 높이 추정치 제공
- 행 선택시 하고 싶은 작업
- 스와이프시 작업
- 테이블 내용 편집 지원

과제 : 2개
메서드가 하는 일



```
func tableView(UITableView, willSelectRowAt: IndexPath) -> IndexPath?
```

지정된 행을 선택하려고 대리인에게 알립니다.

```
func tableView(UITableView, didSelectRowAt: IndexPath)
```

지정된 행이 이제 선택되었음을 대리인에게 알립니다.

```
func tableView(UITableView, willDeselectRowAt: IndexPath) -> IndexPath?
```

지정된 행을 선택 해제하려고한다는 것을 대리인에게 알립니다.

```
func tableView(UITableView, didDeselectRowAt: IndexPath)
```

지정된 행이 선택 해제되었음을 델리게이트에게 알립니다.

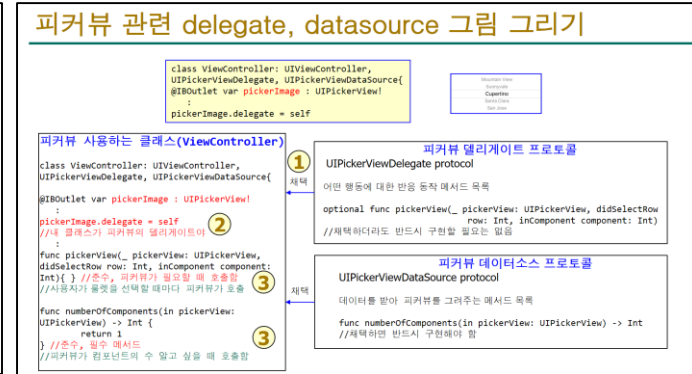
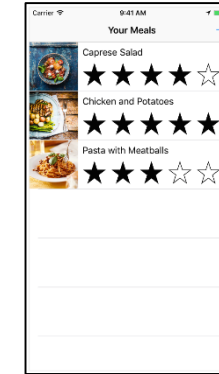
과제: 테이블뷰 관련 delegate, datasource 그림 그리기

■ UITableViewDelegate

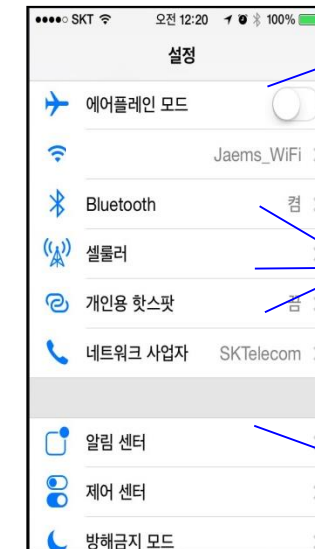
- <https://developer.apple.com/documentation/uikit/uitableviewdelegate>

■ UITableViewDataSource

- <https://developer.apple.com/documentation/uikit/uitableviewdatasource>



```
class ViewController: UIViewController, UITableViewDelegate, UITableViewDataSource {
    @IBOutlet var tableView: UITableView!
    :
}
```



섹션

행(row)
셀(cell)

섹션

UIViewController와 UITableViewController

- UITableView클래스를 상속받는다면?
- <https://developer.apple.com/documentation/uikit/uitableview>

```
class ViewController: UITableViewController {  
}
```

```
class ViewController1: UIViewController, UITableViewDelegate, UITableViewDataSource{  
}
```

과제 : 위의 ViewController1 클래스를 extension으로 나누기

아래처럼 소스를 작성하면 클래스가 너무 비대해짐

```
class ViewController: UIViewController, UIPickerViewDelegate, UIPickerViewDataSource{
```

소스를 다음과 같이 나누면 깔끔하고 이해하기 쉬움

```
class ViewController: UIViewController{  
extension ViewController: UIPickerViewDelegate{  
extension ViewController: UIPickerViewDataSource{
```

열거형 (enum)

열거형(enum)

- <https://docs.swift.org/swift-book/documentation/the-swift-programming-language/enumerations/>
- 관련있는 데이터들이 멤버로 구성되어 있는 자료형 객체
 - 원치 않는 값이 잘못 입력되는 것 방지
 - 입력 받을 값이 한정되어 있을 때
 - 특정 값 중 하나만 선택하게 할 때
- 색깔
 - 빨강, 녹색, 파랑
- 성별
 - 남, 여

열거형 정의

- enum 열거형명{
 열거형 정의
}
- enum Planet {
 case Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune
} //하나의 case문에 멤버들 나열하는 것도 가능
- enum Compass {
 case North
 case South
 case East
 case West
}
- //var x : Compass //Compass형 인스턴스 x
- print(Compass.North) //North
- var x = Compass.West //print(type(of:x)), Compassd
- x = .East //print(x), East
 - 문맥에서 타입의 추론이 가능한 시점 (등호 좌변의 변수 타입이 확정적일 때)에는 열거형명 생략 가능

열거형 멤버별 기능 정의

```
enum Compass {  
    case North  
    case South  
    case East  
    case West  
}  
  
var direction : Compass  
direction = .South  
switch direction { //switch의 비교값이 열거형 Compass  
case .North:      //direction이 .North이면 "북" 출력  
    print("북")  
case .South:  
    print("남")  
case .East:  
    print("동")  
case .West:  
    print("서") //모든 열거형 case를 포함하면 default 없어도 됨  
}
```

swift 열거형 멤버에는 메서드도 가능

```
enum Week: String {
    case Mon, Tue, Wed, Thur, Fri, Sat, Sun
    func printWeek() { //메서드도 가능
        switch self {
            case .Mon, .Tue, .Wed, .Thur, .Fri:
                print("주중")
            case .Sat, .Sun:
                print("주말")
        }
    }
}

Week.Sun.printWeek() //??
```

열거형의 rawValue

```
enum Color: Int {  
    case red  
    case green = 2  
    case blue  
}  
print(Color.red)    //red  
print(Color.blue)  
print(Color.red.rawValue) //0  
print(Color.blue.rawValue)
```

String형 값을 갖는 열거형의 rawValue

```
enum Week: String {  
    case Monday = "월"  
    case Tuesday = "화"  
    case Wednesday = "수"  
    case Thursday = "목"  
    case Friday = "금"  
    case Saturday //값이 지정되지 않으면 case 이름이 할당됨  
    case Sunday   // = "Sunday"  
}  
  
print(Week.Monday) //Monday  
print(Week.Monday.rawValue) //월  
print(Week.Sunday)  
print(Week.Sunday.rawValue)
```

연관 값(associated value)을 갖는 Enum

```
enum Date {  
    case intDate(Int, Int, Int)  //(int, Int, Int)형 연관값을 갖는 intDate  
    case stringDate(String)      //String형 연관값을 갖는 stringDate  
}  
  
var todayDate = Date.intDate(2023, 4, 30)  
todayDate = Date.stringDate("2023년 5월 20일") //주식처리하면?  
switch todayDate {  
    case .intDate(let year, let month, let day):  
        print("\(year)년 \(month)월 \(day)일")  
    case .stringDate(let date):  
        print(date)  
}
```

옵셔널은 연관 값(associated value)을 갖는 enum

```
let age: Int? = 30 //Optional(30)
```

```
switch age {  
case .none: // nil인 경우  
    print("나이 정보가 없습니다.")  
case .some(let a) where a < 20:  
    print("\(a)살 미성년자입니다")  
case .some(let a) where a < 71:  
    print("\(a)살 성인입니다")  
default:  
    print("경로우대입니다")  
} //30살 성인입니다.
```

```
public enum Optional<Wrapped> {  
    case none  
    case some(Wrapped)  
}
```

```
var x : Int? = 20 //some(20)  
var y : Int? = Optional.some(10)  
var z : Int? = Optional.none  
var x1 : Optional<Int> = 30  
print(x, y, z, x1) //??
```

구조체 (struct)

구조체

- <https://docs.swift.org/swift-book/documentation/the-swift-programming-language/classesandstructures/>
- Memberwise Initializer가 자동으로 만들어짐
- Int, Double, String 등 기본 자료형은 구조체
 - <https://developer.apple.com/documentation/swift/int>
 - @frozen struct Int
 - @frozen attribute: 저장 property 추가, 삭제 등 변경 불가
 - <https://docs.swift.org/swift-book/documentation/the-swift-programming-language/attributes/>
 - Array, Dictionary, Set은 Generic Structure
 - <https://developer.apple.com/documentation/swift/array>
 - @frozen struct Array<Element>
- 구조체/enum의 인스턴스는 값(value) 타입, 클래스의 인스턴스는 참조(reference) 타입
- 구조체는 상속 불가

구조체 : Memberwise Initializer 자동 생성

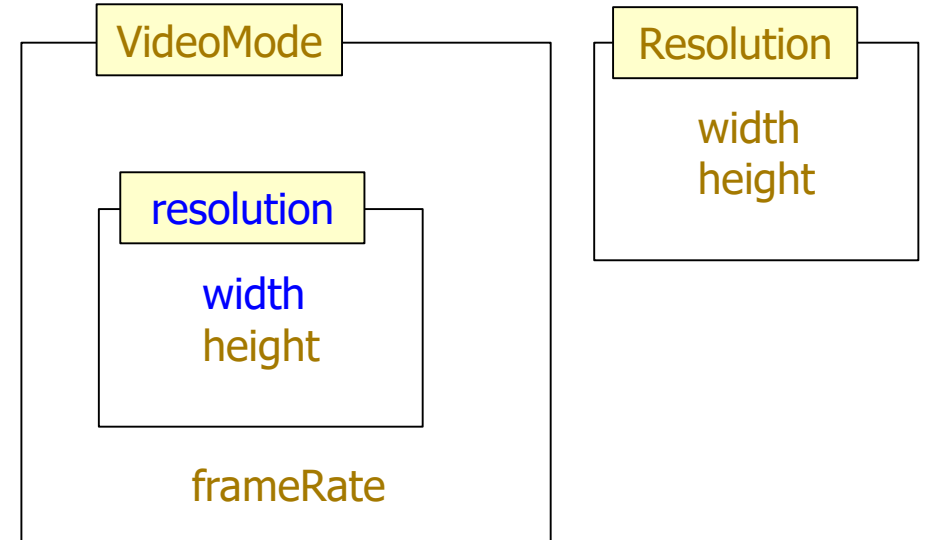
```
struct Resolution { //구조체 정의
    var width = 1024 //프로퍼티
    var height = 768
}
let myComputer = Resolution() //인스턴스 생성
print(myComputer.width) //프로퍼티 접근
```

```
struct Resolution { //구조체 정의
    var width : Int //프로퍼티 초기값이 없어요!!
    var height : Int
} //init()메서드 없어요, 그런데!
let myComputer = Resolution(width:1920,height:1080) //Memberwise Initializer
print(myComputer.width)
```

■ 과제 struct을 class로 변경하면 오류. init만들어 오류 제거하기

클래스 내에 구조체

```
struct Resolution {  
    var width = 1024  
    var height = 768  
}  
  
class VideoMode {  
    var resolution = Resolution()  
    var frameRate = 0.0  
}  
  
let myVideo = VideoMode()  
print(myVideo.resolution.width)
```



Swift 기본 데이터 타입은 모두 구조체

- `public struct Int`
- `public struct Double`
- `public struct String`
- `public struct Array<Element>`
 - `public` : 모듈(앱)의 모든 소스 파일 내에서 사용할 수 있으며, 정의한 모듈을 가져오는 다른 모듈의 소스파일에서도 사용 가능

클래스 vs. 구조체 vs. 열거형

클래스(class)와 구조체(structure)의 공통점

- <https://docs.swift.org/swift-book/documentation/the-swift-programming-language/classesandstructures/>
- Define **properties** to store values
 - Define **methods** to provide functionality
 - Define **subscripts** to provide access to their values using subscript syntax
 - Define **initializers** to set up their initial state
 - Be **extended** to expand their functionality beyond a default implementation
 - Conform to **protocols** to provide standard functionality of a certain kind
 - 프로퍼티(property)를 정의할 수 있음
 - 메서드를 정의할 수 있음
 - []를 사용해 첨자(subscript) 문법으로 내부의 값을 액세스할 수 있는 첨자를 정의할 수 있음
 - 클래스, 구조체, 열거형이 컬렉션 멤버에 접근하기 위한 문법
 - 초기 상태 설정을 위한 초기화 함수(initializer)를 정의할 수 있음
 - extension 가능
 - extension Double {}
 - protocol 채택 가능
 - class Man : Runnable {}

```
struct Num{  
    let num = [1, 2, 3]  
    subscript(i:Int) -> Int{  
        return num[i]  
    }  
}  
  
var n = Num()  
print(n)  
print(n[0], n[1], n[2])
```

class가 struct보다 더 갖는 특징

- 상속이 가능
 - 타입 캐스팅(is as as? as!)을 통해 실행 시점에 클래스 인스턴스의 타입을 검사하고 해석 가능
 - deinitializer(deinit{ })로 사용한 자원을 반환 가능
 - 참조 카운팅을 통해 한 클래스 인스턴스를 여러 곳에서 참조(사용) 가능
-
- Inheritance enables one class to inherit the characteristics of another.
 - Type casting enables you to check and interpret the type of a class instance at runtime.
 - Deinitializers enable an instance of a class to free up any resources it has assigned.
 - Reference counting allows more than one reference to a class instance.

클래스/구조체 정의하기

- 클래스 정의하기

- class 이름 { ... }

- 구조체 정의하기

- struct 이름 { ... }

- 타입 이름에는 Upper Camel Case를 사용

- 대문자로 시작

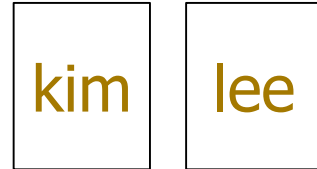
- 클래스/구조체 안의 프로퍼티나 메서드는 lower Camel Case를 사용

```
struct Resolution {  
    var width = 0  
    var height = 0  
}
```

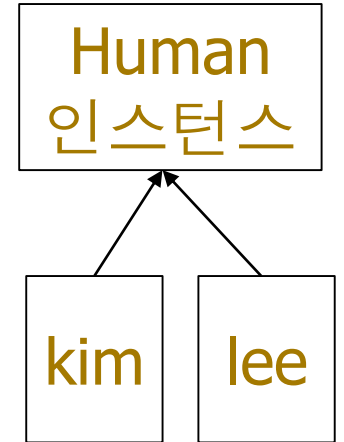
```
class VideoMode {  
    var resolution = Resolution()  
    var interlaced = false  
    var frameRate = 0.0  
    var name: String?  
}
```

구조체는 값 타입(value type) 클래스는 참조 타입(reference type)

```
struct Human {  
    var age : Int = 1  
}  
var kim = Human()  
var lee = kim //값 타입  
print(kim.age, lee.age)  
lee.age = 20  
print(kim.age, lee.age)  
kim.age = 30  
print(kim.age, lee.age)  
var x = 1  
var y = x  
print(x,y)  
x = 2  
print(x,y)  
y = 3  
print(x,y)  
//값 타입은 복사할 때 새로운 데이터가 하나 더 생김
```



```
class Human {  
    var age : Int = 1  
}  
var kim = Human()  
var lee = kim //참조 타입  
print(kim.age, lee.age)  
lee.age = 20  
print(kim.age, lee.age)  
kim.age = 30  
print(kim.age, lee.age)
```



//참조 타입은 복사할 때 주소를 복사해서
//한 데이터의 reference가 2개 생김

구조체는 값 타입(value type) 클래스는 참조 타입(reference type)

```
struct Resolution {  
    var width = 0  
    var height = 0  
}  
class VideoMode {  
    var resolution = Resolution()  
    var frameRate = 0  
    var name: String?  
}  
  
var hd = Resolution(width: 1920, height: 1080)  
           //자동 Memberwise Initializer  
var highDef = hd  
           //구조체는 값타입(value type)  
  
print(hd.width, highDef.width)  
hd.width = 1024  
print(hd.width, highDef.width)
```

```
var xMonitor = VideoMode()  
xMonitor.resolution = hd  
xMonitor.name = "LG"  
xMonitor.frameRate = 30  
print(xMonitor.frameRate)  
  
var yMonitor = xMonitor  
           //클래스는 참조타입(reference type)  
  
yMonitor.frameRate = 25  
  
print(yMonitor.frameRate)  
print(xMonitor.frameRate)
```

언제 클래스를 쓰고 언제 구조체를 쓰나?

■ 클래스는 참조타입, 구조체는 값 타입

- 구조체는 간단한 데이터 값들을 한데 묶어서 사용하는 경우
- 전체 덩어리 크기가 작은 경우, 복사를 통해 전달해도 좋은 경우 구조체
- 멀티 쓰레드 환경이라면 구조체가 더 안전
- 구조체는 상속할 필요가 없는 경우
 - 너비, 높이를 표현하는 기하학적 모양을 처리할 경우
 - 시작값, 증분, 길이 등으로 순열을 표현할 경우
 - 3차원 좌표 시스템의 각 좌표