Swift Study 15

2016. 03.11

Swift 문법

- 접근제한자 / final / 워크스페이스 생성
- 프로토콜(protocol)
- 델리게이트 패턴 (delegate pattern)

접근 제한자 (access cotrol)

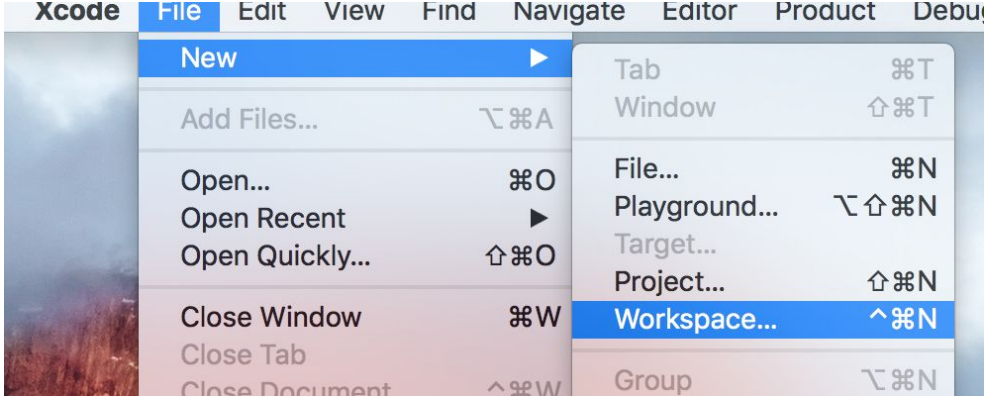
- 클래스(class), 구조체(struct), 열거형(enum), 변수, 함수 등에 대한 접근제한
- swift 3.0 5가지 open, public, private, internal, fileprivate 제공

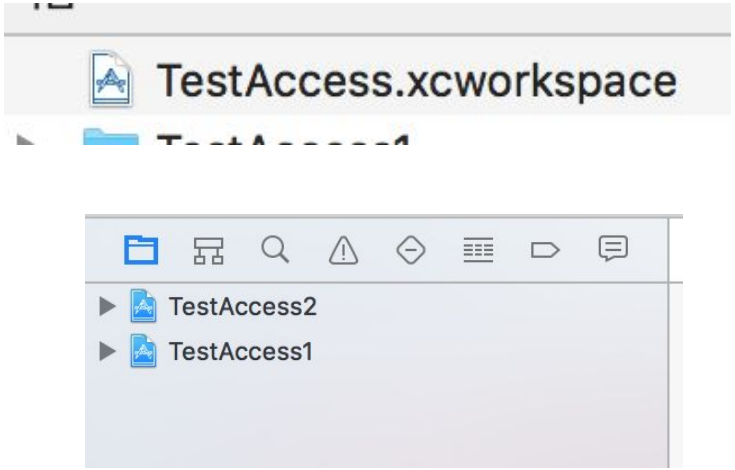
접근 제한자 종류	특징
open	외부모듈에 접근제한이 없으며, 외부모듈에 대한 상속, 메소드 오버라이딩 가능
public	외부모듈에 접근제한이 없으며, 외부모듈에 대한 상속, 메소드 오버라이딩 불가능
internal	해당 모듈만 접근가능 (멀티 프로젝트가 아닌 경우 **public** 동급)
fileprivate	동일 파일 내에서만 접근가능
private	블록({ }) 안에서도 제한적으로 접근가능

***final** : 특정 프로퍼티 / 메소드의 **overriding**을 막는 키워드

워크스페이스 생성

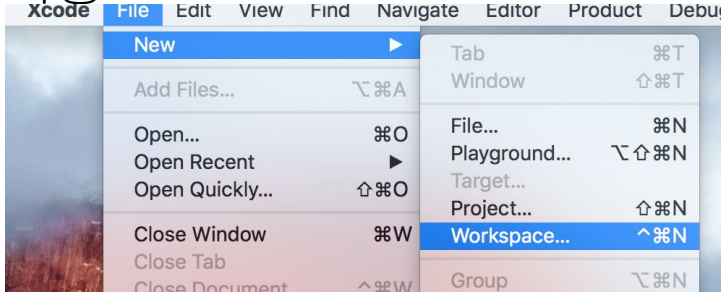
- 여러 프로젝트(모듈)를 묶는 큰 그룹 개념
- 각 프로젝트는 모듈로 취급 (import시 프로젝트명)

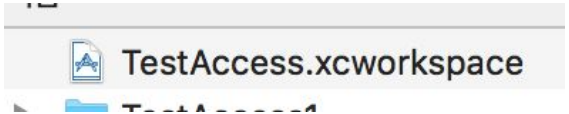


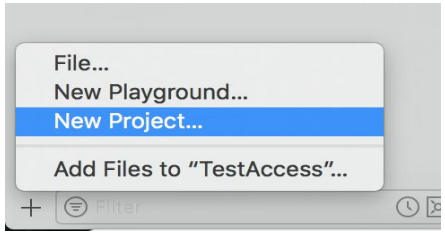


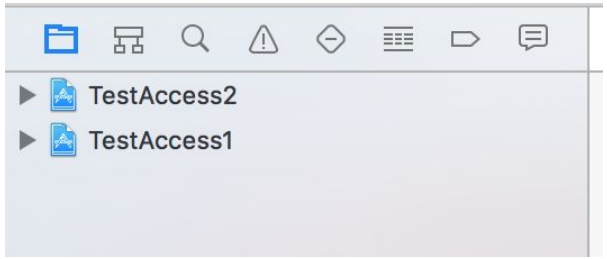
워크스페이스 생성

- 여러 프로젝트(모듈)를 묶는 큰 그룹 개념
- 각 프로젝트는 모듈로 취급 (import시 프로젝트명)
- 네이게이션 영역 왼쪽 하단의 + 눌러서 필요한 프로젝트 생성 및 추가 가능









프로토콜 (protocol)

- 구현체 없는 함수/메소드 (함수이름과 매개변수/반환형만 정의된 형태)
- 메소드에 대한 설계/명세의 목적으로 사용되는 문법 (프로퍼티/초기화 정의가능)
- type으로 취급되기 때문에 변수 대입, return값, 매개변수 등 사용가능
- protocol이라는 키워드를 쓰며, 클래스 / 구조체 / 열거형 뒤에 " : " 으로 프로토콜 채택 (adopt)

<형식>

```
protocol 프로토콜명 {
    구현해야 할 메소드 정의1
    구현해야 할 메소드 정의2
}

class/struct/enum 객체명 : 프로토콜명 {

}
```

예제)

```
protocol TestProtocol {
    func stringTest() -> String
    func addOne(num: Int) -> Int
}

class/struct/enum Test : TestProtocol {
    func printTest() -> String {
        return "test"
    }

    func addOne(num: Int) -> Int {
        return num+1
    }
}
```

프로토콜 - 프로퍼티 정의

- 프로토콜에서 프로퍼티 정의는 get set 키워드로 읽기전용 / 읽기,쓰기 전용 구분
- get set 경우는 상수 프로퍼티나 읽기전용 연산 프로퍼티를 정의할 수 없음
- get 경우 모든 경우의 프로퍼티 정의 가능

<형식>

```
protocol 프로토콜명 {
    var 프로퍼티명:타입 { get set }
    var 프로퍼티명:타입 { get }
}

class 객체명 : 프로토콜명 {
    var 프로퍼티:타입

    var 프로퍼티: 타입 {
        get {   return 값   }
        set(매개변수) {  //실행구문 }
    }
}
```

예제)

```
protocol TestProtocol {
    var name1:String {get set}
    var description:String {get}
}

class Test : TestProtocol {
    var name1:String
    var description:String {
        get { return "test" }
        set(value) { self.name1 }
    }
}
```

프로토콜 - 메소드 정의

- 메소드는 구현체 없는 몸체만 정의
- 내부프로퍼티 참조하는 mutating 키워드 / 인스턴스 없이 사용가능한 정적(타입) 메소드 정의가능
- 클래스 경우에는 mutating 생략가능 / static키워드 경우 class 키워드로 대체가능

```
protocol Test {
    func testfunc(str:String) -> String                  //general method
    mutating func testfunc2(str:String) -> String   //inner property modify - mutating
    static func testfunc3(str:String) -> String        //static method
}

class Test1 : Test {
        func testfunc(str: String) -> String {  return str }
        func testfunc2(str: String) -> String { self.name1 = "test1234"; return str; }
        static func testfunc3(str: String) -> String {  return str+" static" }
        //class func testfunc3(str: String) -> String {  return str+" static" }
}
```

프로토콜 - 초기화 정의

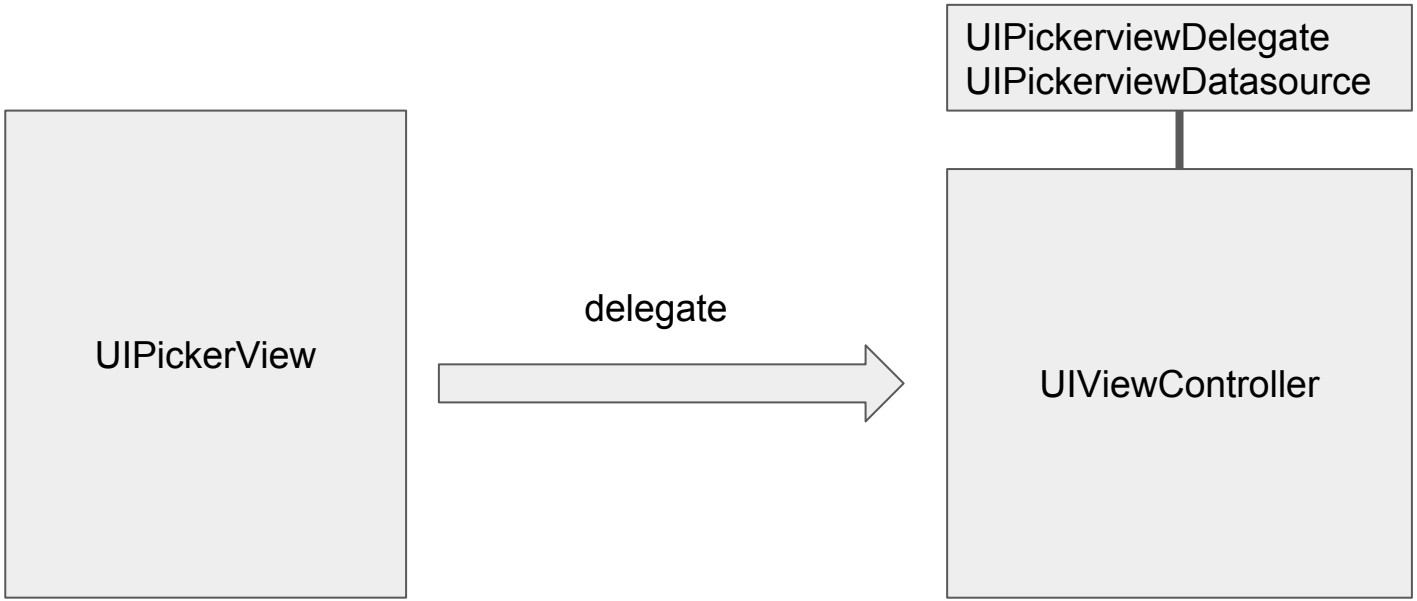
- class / struct / enum에 사용하는 init() 초기화 사용가능
- class 경우 클래스 내부의 init()과 구분하기 위해 프로토콜의 init() 정의시 require 키워드 붙음
- init?() failable initializer 정의가능 (초기화 조건 만족 안할시 옵셔널 특정 초기화 설정)

```
protocol Test {
    init(str:String)                        //initializer
    init?(str:String?, str2:String?)    //failable initializer
}

class Test1 : Test {
        required init(str: String) { self.name1 = str }
        required init?(str: String?, str2: String?) {
                if let s1 = str, let s2 = str2 {
                        self.name1 = s1+s2
                }else{
                        return nil
                }
        }
}
```

델리게이트 패턴 (delegate pattern)

- 특정 대상에게 자신의 임무/기능을 대신 맡기겠금 하는 방식 (대리자 위임)
- cocoa framework의 근간이 되는 이벤트 및 다양한 기능들이 델리게이트 패턴으로 구현되어 있음
- protocol 타입의 delegate / datasource 을 구현하여 대리자가 대신 기능을 구현



델리게이트 패턴 (delegate pattern)

기능을 갖는 클래스

```
protocol UIActionDelegate {
    func click(uiAction: UIAction)
}

class UIAction {
    var delegate: UIActionDelegate?

    func clickEvent() {
        self.delegate?.click(self)
    }
}
```

대신 기능을 구현하는 대리자 클래스

```
class Test: UIActionDelegate {
    let uiAction = UIAction()

    init() {
        self.uiAction.delegate = self
        self.uiAction.clickEvent()
    }

    func click(uiAction: UIAction) {
        //해당 메소드에 대한 기능 구현....
    }
}
```

델리게이트 패턴

```
import UIKit

class ViewController: UIViewController, UIPickerViewDelegate, UIPickerViewDataSource  {

    override func viewDidLoad() {
        super.viewDidLoad()

        pickerView.delegate = self
        pickerView.dataSource = self
    }

    //열의 갯수
    func numberOfComponents(in pickerView: UIPickerView) -> Int {
        return PICKER_VIEW_COLNUM
    }

    //행의 갯수(실제 보일 컨텐츠 갯수)
    func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
        return imageFileName.count
    }

    //피커뷰의 높이
    func pickerView(_ pickerView: UIPickerView, rowHeightForComponent component: Int) -> CGFloat {
        return PICKER_VIEW_HEIGHT
    }

    //피커뷰 이름 타이틀 세팅
    func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) -> String? {
        return imageFileName[row]
    }
```

델리게이트 패턴

```
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

    @available(iOS 2.0, *)
    optional public func pickerView(_ pickerView: UIPickerView, viewForRow row: Int, forComponent component: Int, reusing view: UIView?) -> UIView


    @available(iOS 2.0, *)
    optional public func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int)
}
```

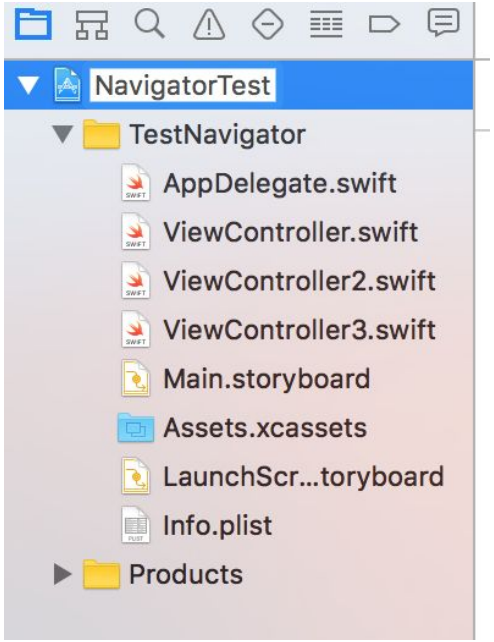
ios

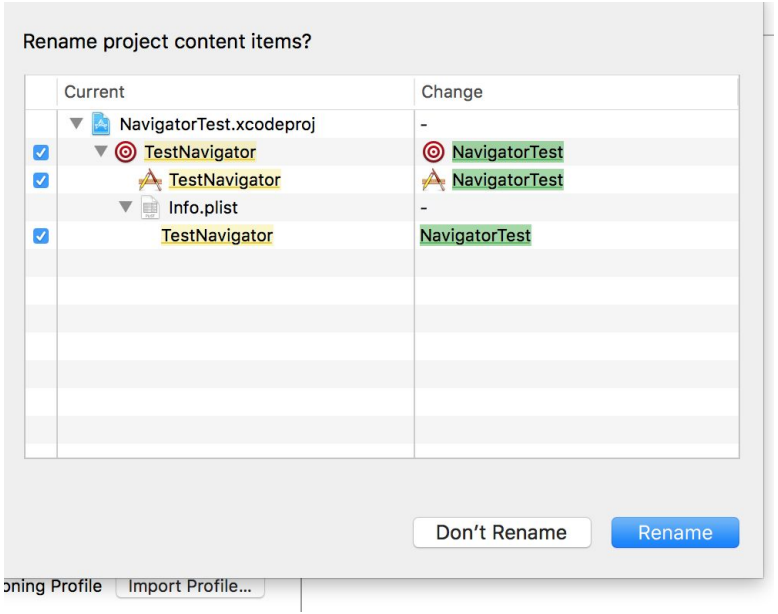
- 프로젝트명 수정방법

- 화면 전환 (프로그래밍 방식 / segue 방식) / unwind

프로젝트명 수정

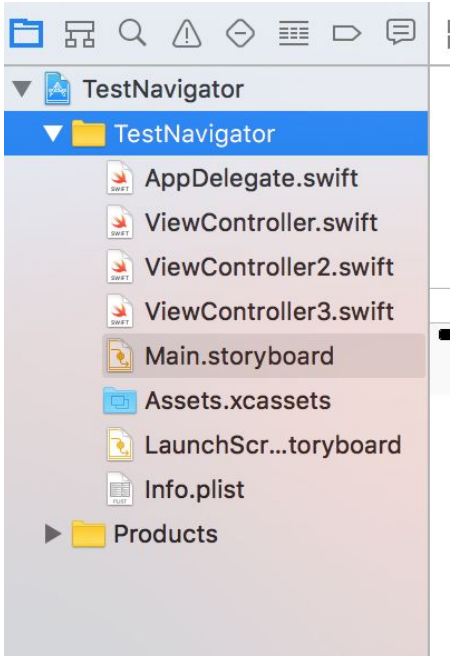
- 네이게이터 영역의 프로젝트명 폴더를 클릭하여 이름 수정

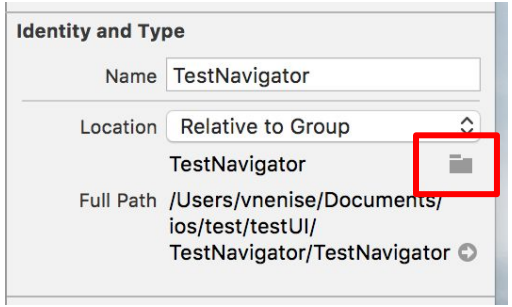


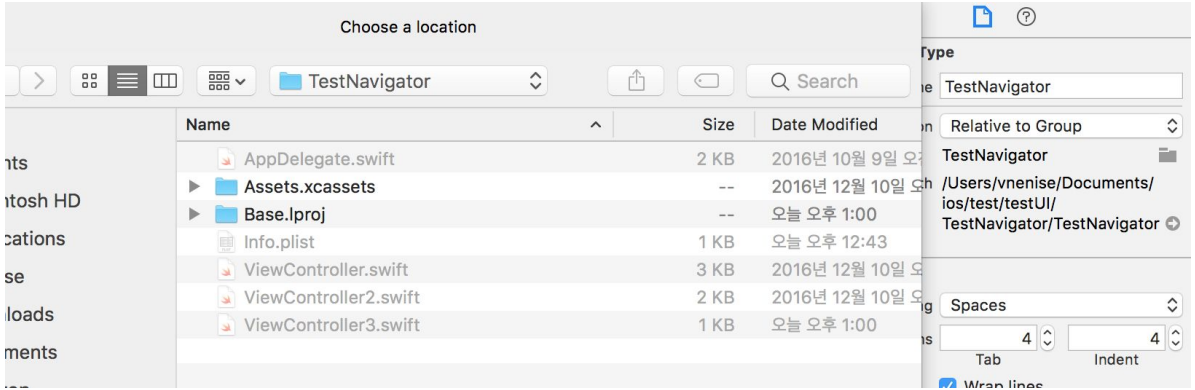


프로젝트명 수정

- 프로젝트명 수정 후에는 실제 참조하는 프로젝트 폴더의 위치 설정 필요.

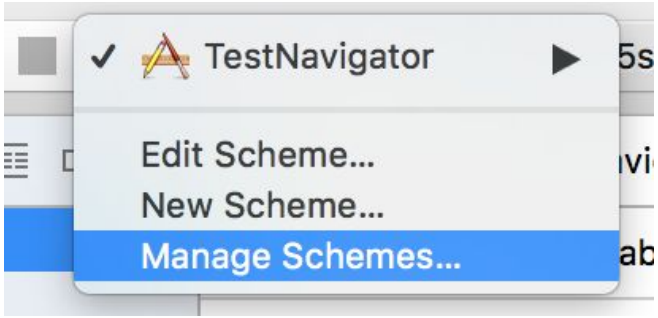


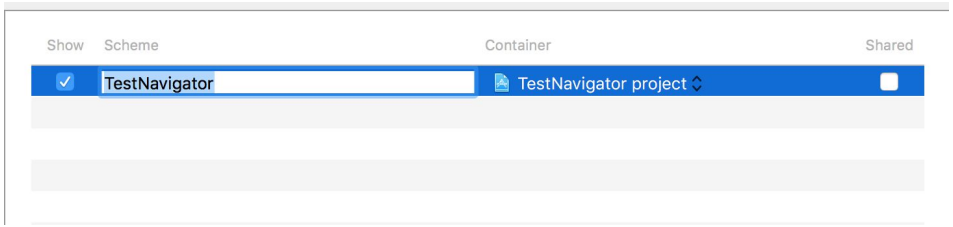




프로젝트명 수정

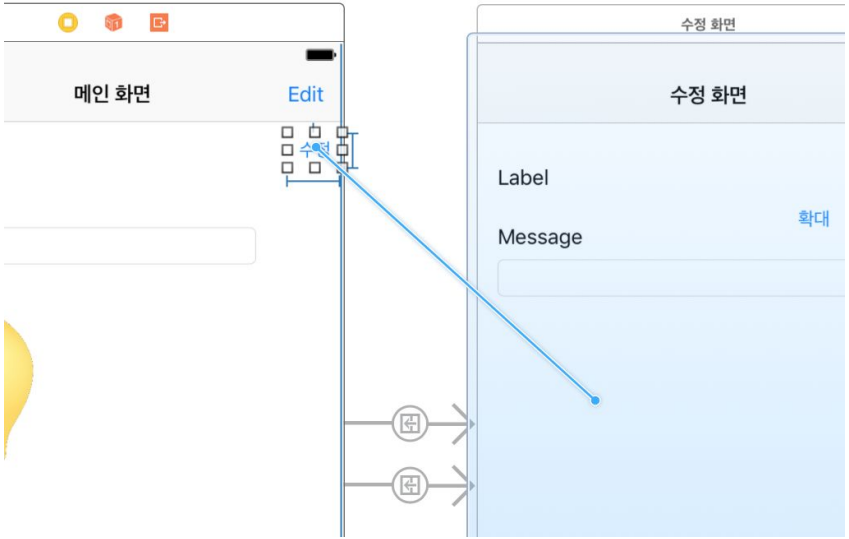
- 실행앱 이름 변경하고 싶을시 manage Schemes에 Scheme명 변경

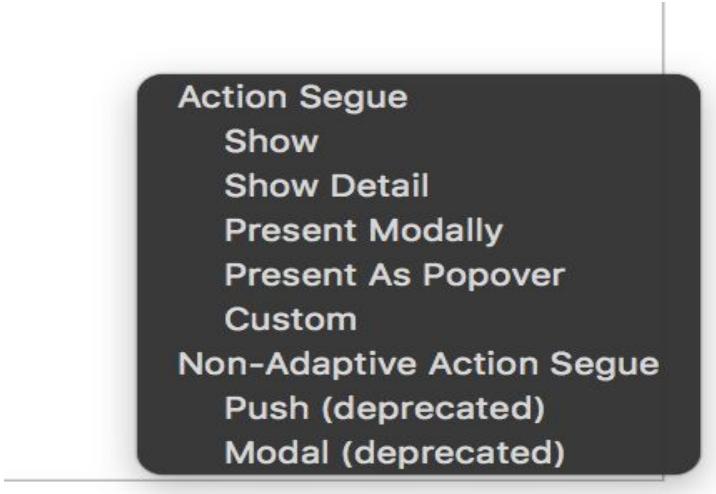




Segue

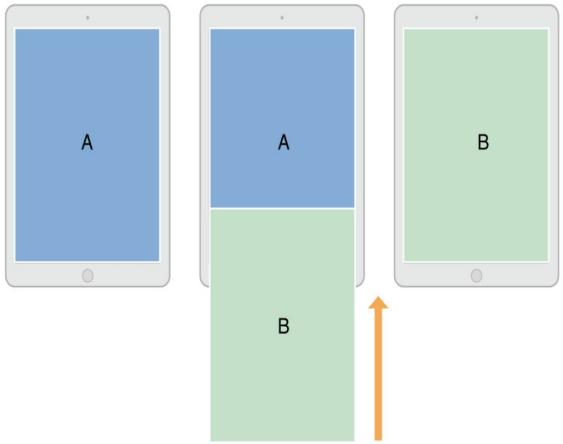
- 화면전환에 대한 식별자 (안드로이드: intent와 유사)
- 화면전환에는 직접 UI에 대한 presentController를 호출하거나 Segue 식별자로 전환방식
- IB(interface builder)에는 segue에 대한 화면전환 action 제공
- UIController (manual segue) 또는 UIControl(action segue) 형태의 segue 나눔

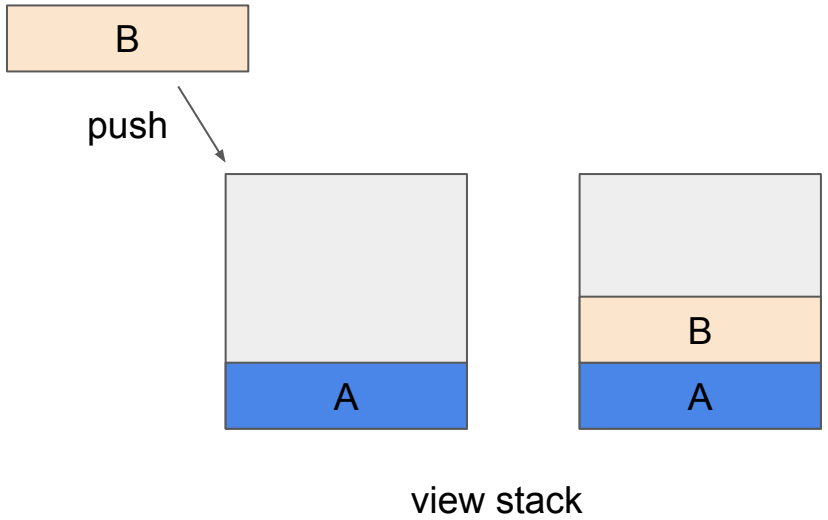




Segue - Show (push)

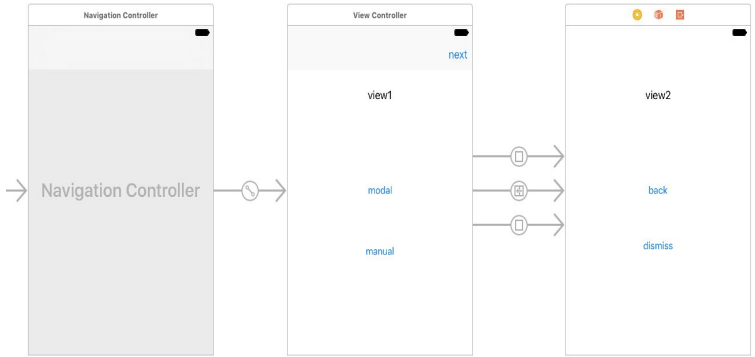
- NavigationViewController에 대한 뷰스택에 쌓으면서 보여주는 화면형태 (show)
- NavigationViewController는 push/pop 구조로 화면전환 관리
  (pushViewController / popViewController)

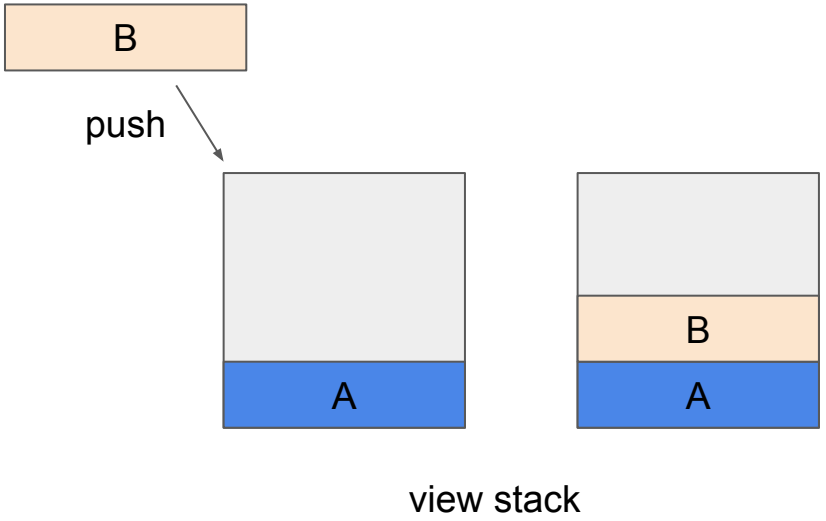




NavigationViewController

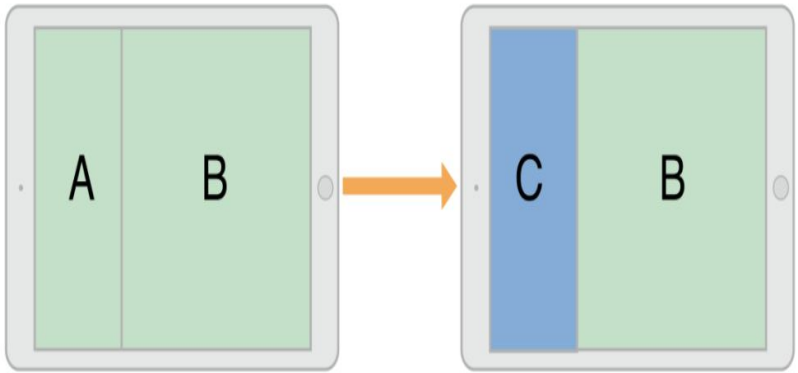
- 최초 실행 ViewController를 rootViewController 지칭
- 스택구조로 뷰를 관리되어, 화면전환시 push/pop 메소드로 전환 필요
- NavigationViewController 연결되어 상단의 navigation 메뉴가 자동 할당됨





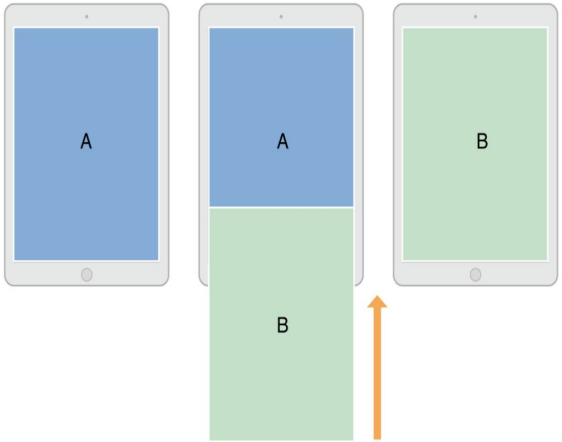
Segue - Show Detail (replace)

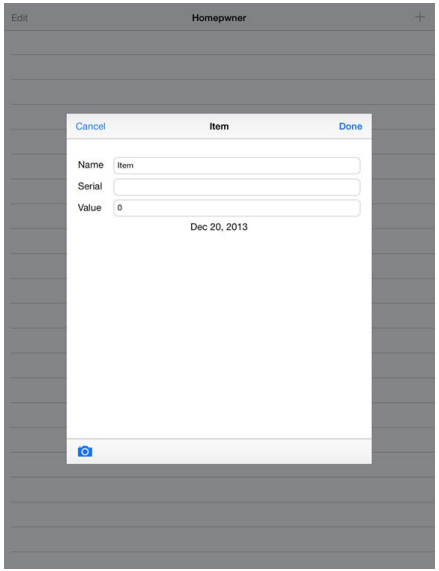
- master와 detail로 나눠지는 화면구성에서 detail 영역을 대치(replace)해서 화면형태
- 태블릿이 지원되는 Universal 앱 경우 show-detail 화면 많이 활용
- view스택에서는 영향이 없음



Segue - Present Modally

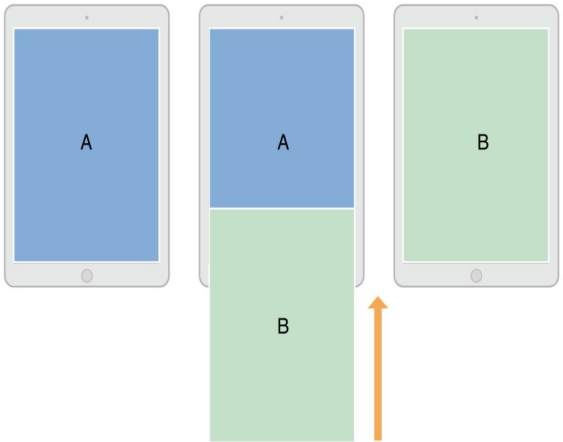
- 기존 화면을 덮으면서 위로 뜨는 화면형태 (ios에서 가장 일반적인 뷰 전환)
- ios 8부터 modal 대신 present modally 사용 (modal deprecated)





presenting / presentedViewController

- present() 화면전환 경우 두 뷰 간의 관계를 presented / presenting으로 구별
- 보여진 뷰를 presentedViewController에 참조 포인터 저장
- 보여지고 있는 뷰는 presentingViewController에 참조 포인터 저장

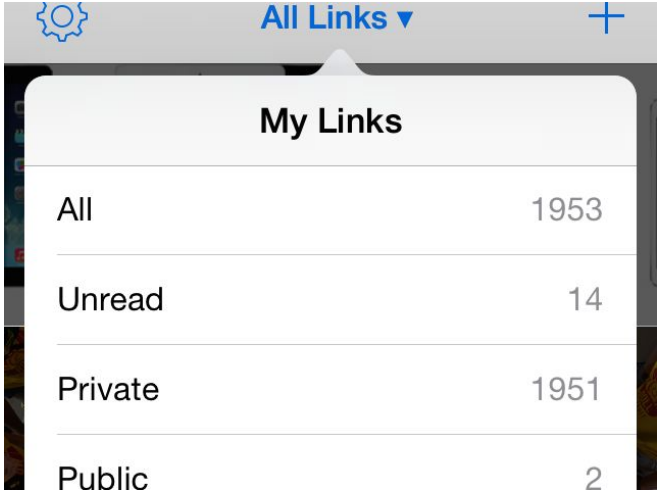


A presentedViewController

B presentingViewController

Segue - Present As Popover

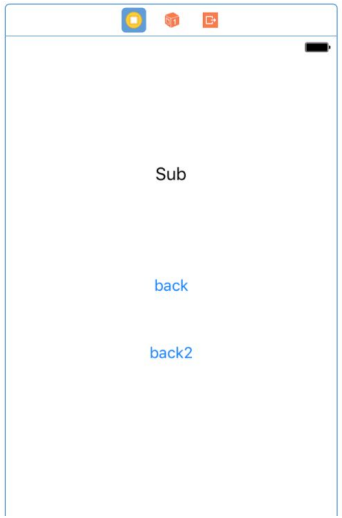
- 작은 팝업형태의 뷰 띄우는 화면형태
- 새로 띄운 뷰의 바깥영역을 터치하면 뷰 사라짐

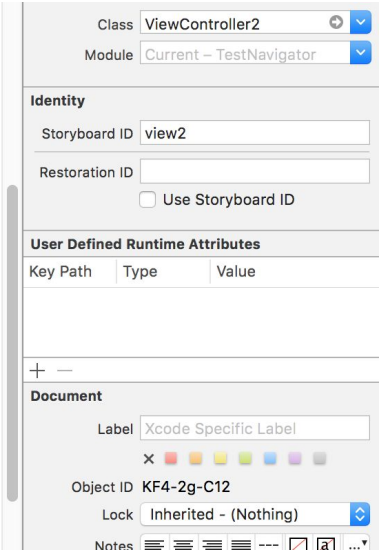


화면전환 - 프로그래밍 방식

- 특정 ViewController 인스턴스 생성하여 화면전환
- ViewController에 스토리보드 특정id 부여

1) present()를 통한 present modally 방식
2) NavigationViewController를 통한 pushViewController 방식






```
let uv = self.storyboard?.instantiateViewController(withIdentifier: "view2")

self.present(uv, animated: true)       // modally 방식
self.navigationController?.pushViewController(uv, animated: true) //nvaigation 방식
```

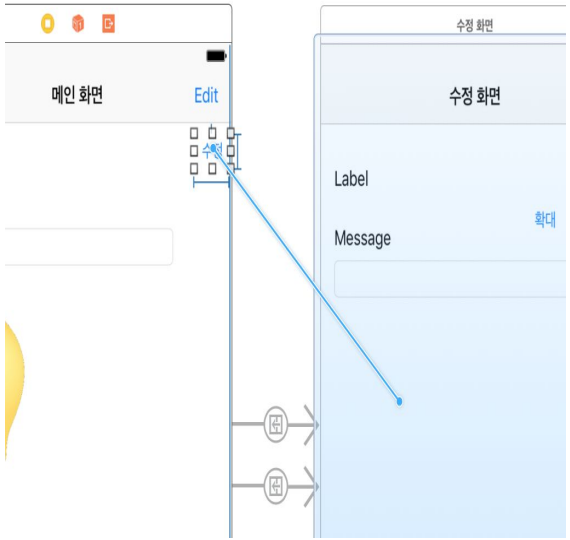
Segue - 프로그래밍 방식

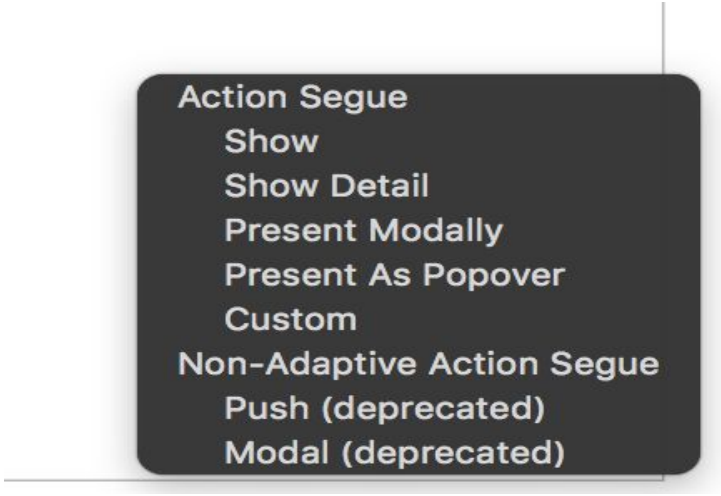
- 이벤트 함수에 직접 불러온 ViewController 지정 후 presentController 함수호출

```
@IBAction func segueSend(_ sender: UIButton) {
        let storyboard = UIStoryboard(name: "Main", bundle: nil)
        let view2 = storyboard.instantiateViewController(withIdentifier: "ViewController2") as! ViewController2
        present(view2, animated: true, completion: nil)
}
```

화면전환 - segueway 이용

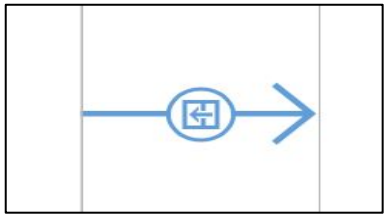
- Segue라는 화면전환 객체를 이용한 방식
- UIControl 상속받고 있는 특정대상에 화면전환 이벤트 거는 방식 (action segue)
- UIViewController 자신에게 화면전환 이벤트 거는 방식 (manual segue)

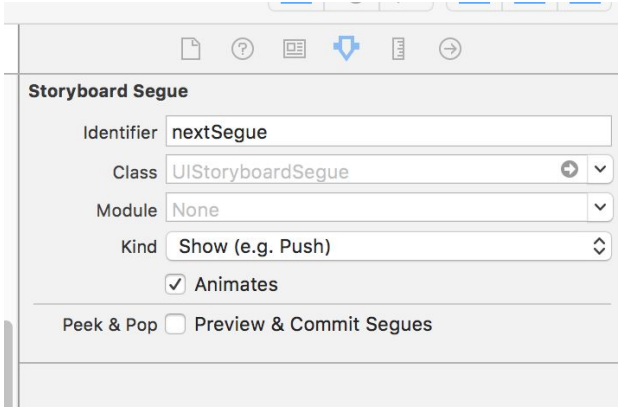




Segue - 제어방법

- segue에 identifier 설정 후 ViewController에서 segue 관련 code정의





func **performSegue**(**withIdentifier** identifier: String, sender: Any?) **//생성된 특정 segue 호출**

func **prepare**(**for segue**: UIStoryboardSegue, sender: Any?) **//segue가 실행전에 초기화 또는 특정기능 설정**

UIViewController 자신.**performSegue**(withIdentifier: "세그 identifier id", sender: 이벤트대상) **//특정이벤트에서segue 설정 (manual segue)**

Segue - 제어방법

```
func prepare(for segue: UIStoryboardSegue, sender: Any?)    //segue가 실행전에 초기화 또는 특정기능 설정
```

ViewController.class

```
func prepare(for segue: UIStoryboardSegue, sender: Any?) {

        let editViewController = segue.destination as! EditViewController

        if segue.identifier == "nextSegue" {
            editViewController.textWayValue = "segue : use button"
        }
}
```

Segue - dismiss / unwind

- segue간의 연결고리가 되어 있어 dismiss()나 unwind기능으로 이전화면 전환
- UIViewController 에 dismiss()함수 내장
- navigationView를 상속받는 화면은 popViewController 사용

```
self.dismiss(animated: true, completion: nil)

self.presentingViewController?.dismiss(animated: true, completion: nil)

self.navigationController?.popViewController(animated: true)
```