스터디 2주차

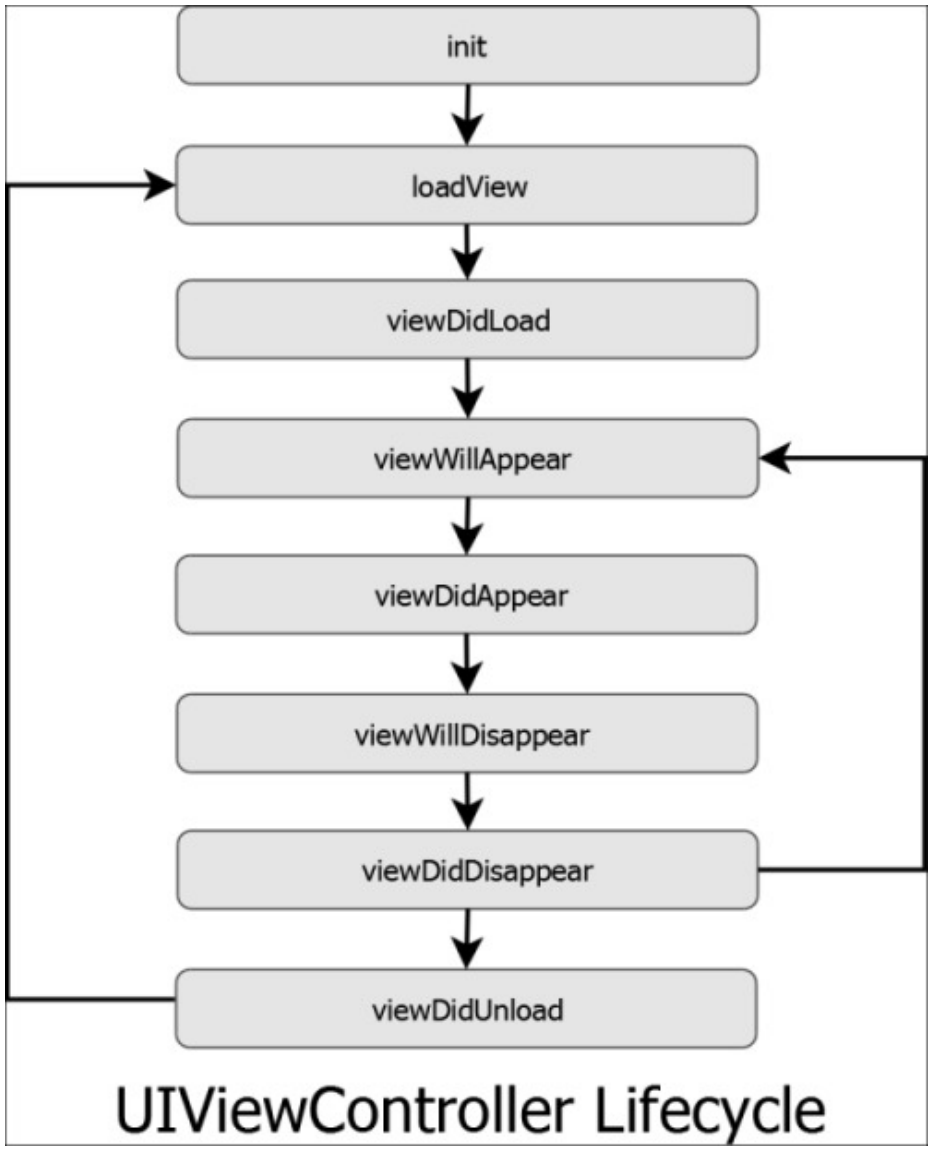
- 재혁

iOS App Lifecycle
UITextView
UITextField

- 지호

iOS viewController LifeCycle
UILabel
UIButton

View Controller 생명주기



1. viewDidLoad

```swift
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view,
            typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


}
```

프로젝트를 새로 만들면 보이는 ViewController.swift에 있는 viewDidLoad() 함수

viewDidLoad 메서드는 '뷰의 로딩이 완료됐을 때' 시스템에 의해 자동으로 호출된다. 이 메서드가 호출되는 시점에서는 이미 outlet 들이 모두 메모리에 위치하고 있다. 그러므로 사용자에게 화면이 보여지기 전 데이터를 뿌려주는 행위에 대한 코드를 작성할 수 있다. 또한 화면이 로드되기 전 백그라운드에서 처리해주어야하는 작업들이 위치하기 좋은데 그 예로는 네트워크 호출 등이 있다. 일반적으로 리소스를 초기화하거나 초기 화면을 구성하는 용도로 주로 사용한다. 화면이 처음 만들어질 때 한 번만 실행되므로, 처음 한 번만 실행해야 하는 초기화 코드가 있을 경우 이 메소드 내부에 작성하면 된다.

/*
loadView는 컨트롤러가 관리하는 뷰를 '만드는' 역할을 한다. loadView가 뷰를 만들고 메모리에 올린 후에 viewDidLoad가 호출되는 것.
*/

2. viewWillAppear

뷰가 로드 된 후에 viewWillAppear는 뷰가 이제 나타날 거라는 신호를 컨트롤러에게 알리는 역할을 한다. 즉 '뷰가 나타나기 직전'에 호출된다고 볼 수 있다.

- 그렇다면 viewWillAppear과 viewDidLoad의 차이점은?

예를들어 아이폰 설정앱에서 네비게이션 컨트롤러가 쓰여졌는데 자료구조에서의 스택 구조와 같다. push 연산이 있으면 pop 연산도 있는데, 뒤로가기 버튼을 누르면 화면은 사라지는데(pop) 이 때 pop된 데이터는 메모리에서 사라지게 된다.

따라서 앱의 초기화 작업은 viewDidLoad에서 해도 되겠지만 다른 뷰에서 갔다가 다시 돌아오는 상황에 해주고 싶은 처리는 viewWillAppear에서 해주면 된다.

3. viewDidAppear

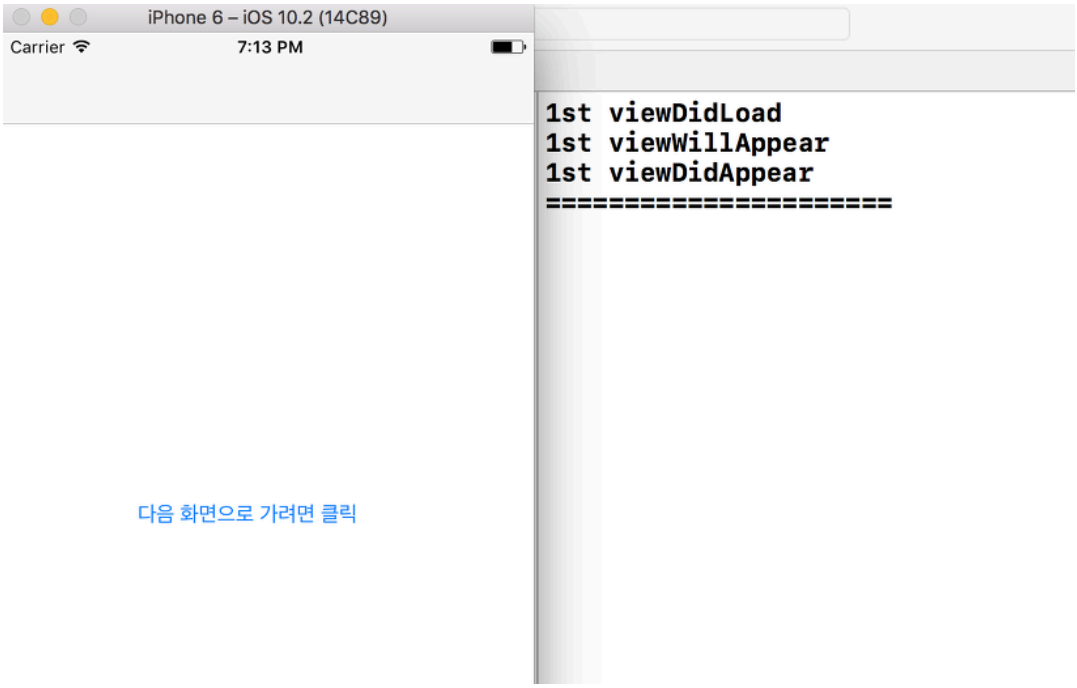
viewDidAppear은 뷰가 나타났다는 것을 컨트롤러에게 알리는 역할을 한다. 또한 화면에 적용될 애니메이션을 그려준다. 이 viewDidAppear는 '뷰가 화면에 나타난 직후'에 실행된다.

4. viewWillDisappear

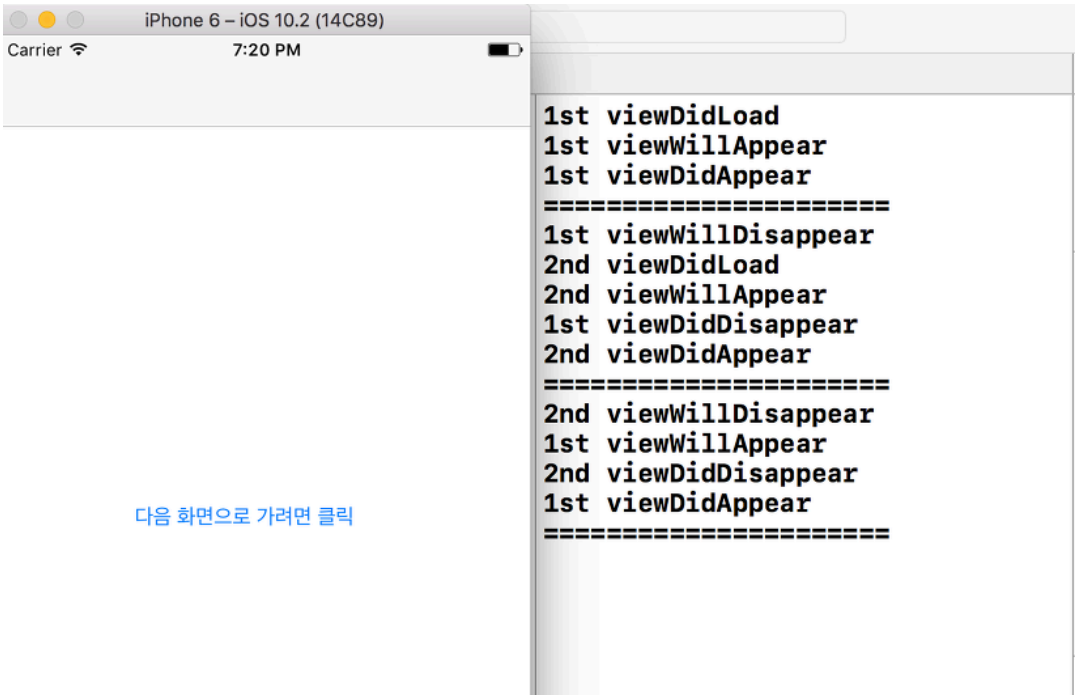
viewWillDisAppear는 '뷰가 사라지기 직전'에 호출되는 함수이다. 뷰가 삭제 되려고 하고있는 것을 뷰 컨트롤러에 통지한다.

5. viewDidDisappear

viewDidDisappear가 호출되면 뷰 컨트롤러가 뷰가 제거되었음을 알려준다. notification을 듣는 행위를 멈추기, 다른 객체의 속성을 observing 하는 것을 멈추기, 디바이스 센서를 점검하거나 네트워크를 호출하는 행위들은 화면이 사라지고 나서는 필요 없는 작업들이기 때문에 적절하다.



두 번째 블록 순서는 iOS 버전에 따라 다른듯 함.



UILabel

문자열을 화면에 보여주고 싶을 때 사용하는 UI.
사용자에게 정적인 정보를 알리기 위해 UIKit에서 제공하는 읽기 전용 텍스트 뷰이다. 글씨 색, 서체 종류, 그림자, 텍스트 정렬 등 간단한 스타일링을 적용할 수 있으며, 한 줄 이상을 출력할 수 있다.

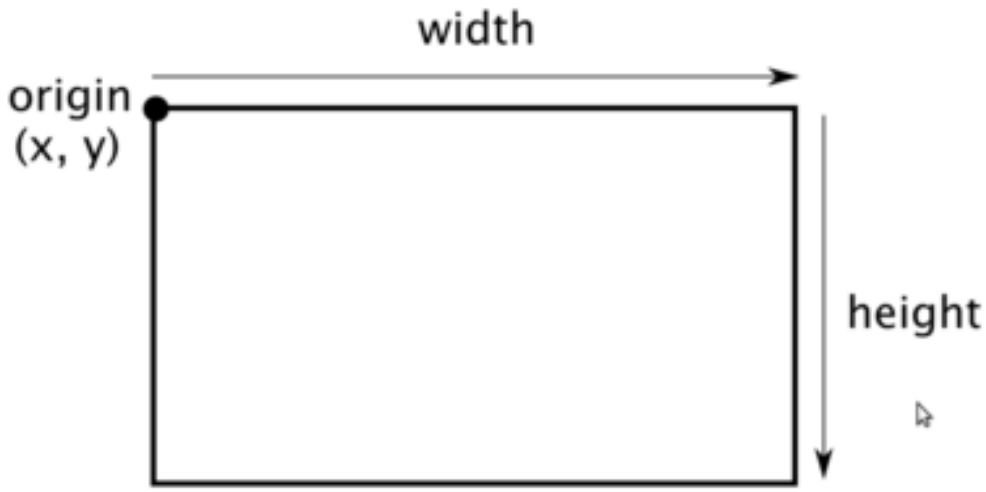
간단한 라벨을 만든 예시
UILabel *label = [[UILabel alloc] initWithFrame:CGRectMake(10, 30, 200, 25)]
[label setText:@“Hi iOS!”]

UILabel의 프로퍼티 및 메소드

1. initWithFrame : UI를 초기화하는 메소드. 위치와 크기를 설정해 초기화한다. CGRect 값으로 설정할 수 있다.

CGRect란? (CG: Core Graphics 프레임워크의 데이터 타입)

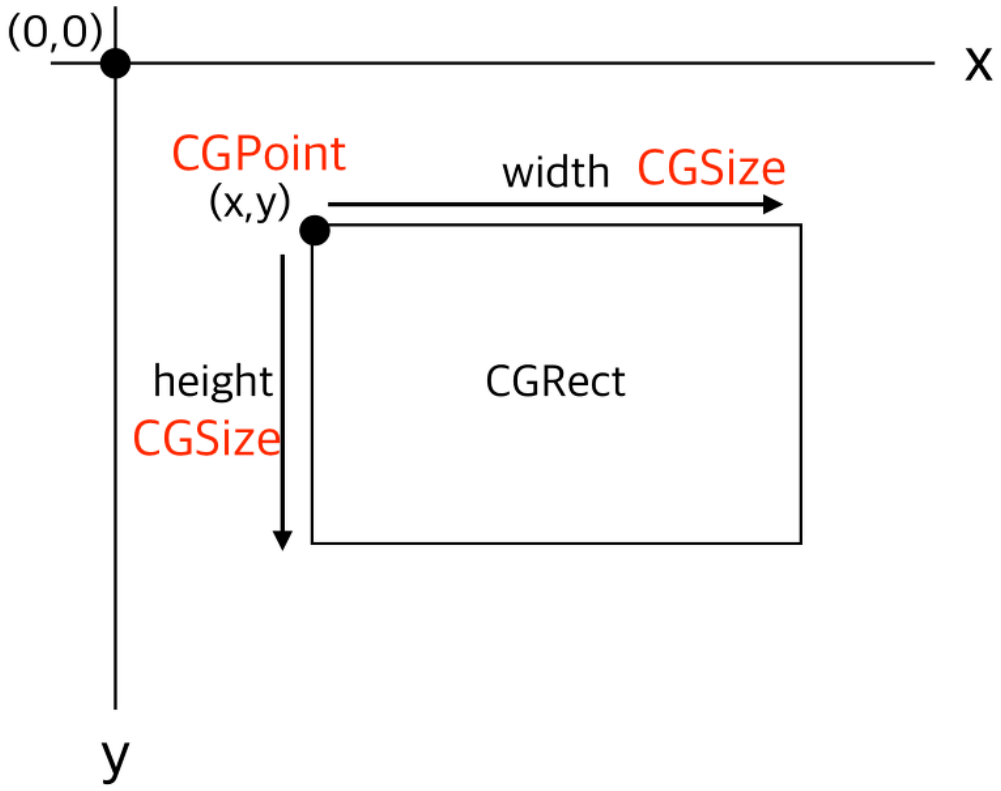
사각형의 위치와 크기를 포함하는 구조체.



너비와 높이를 가지고 있을 뿐만 아니라 원점(origin)을 가지고 있다.

```
public struct CGRect {
    public var origin: CGPoint
    public var size: CGSize
    public init()
    public init(origin: CGPoint, size: CGSize)
}
```

코드상으로 이렇게 구현되어있다.
origin의 타입은 CGPoint, size의 타입은 CGSize.
iOS에서는 위치를 알아야 사각형을 그릴 수 있다.



그 위치를 CGPoint로 나타내고, 크기 즉 너비와 높이는 이를 구조체 변수로 가지는 CGSize로 정하는 것.

```
var rectangle = CGRect(origin: CGPoint(x: 0, y: 0), size: CGSize(width: 50, height: 30))
```

2. alpha: UI의 투명도를 설정하는 프로퍼티. 0~1 사이의 값을 가지며 0은 안 보이게, 1은 투명도 100%로 완전히 보이게 된다.

3. frame: UI의 위치를 설정하는 프로퍼티. CGRect클래스 형태로 설정한다.

4. backgroundColor: UI의 배경색을 설정하는 프로퍼티. UIColor 클래스 형태로 설정한다.

UIColor란?
색상 데이터 및, 때로는 불투명도(알파값)를 저장하는 객체.

5. text: 문자열의 내용을 설정하거나 바꾸고 설정된 값을 가져올 수 있는 프로퍼티.

6. textColor: 문자열의 색깔을 바꾸고 현재 설정된 문자열의 색깔을 가져올 수 있는 프로퍼티. UIColor 클래스 형태로 설정한다.

7. textAlignment: 문자열의 정렬 방식을 선택하는 프로퍼티.
- UITextAlignmentLeft : 문자열을 왼쪽으로 정렬
- UITextAlignmentCenter : 문자열을 가운데로 정렬
- UITextAlignmentRight : 문자열을 오른쪽으로 정렬

8. font: 문자열에 적용할 폰트를 설정하고 가져올 때 사용하는 프로퍼티. UIFont 클래스에서 폰트의 종류와 크기를 설정해 적용할 수 있다.
9. numberOfLines: UILabel에서 표시할 줄의 개수를 설정하고 가져올 때 사용하는 프로퍼티. 기본값은 1.

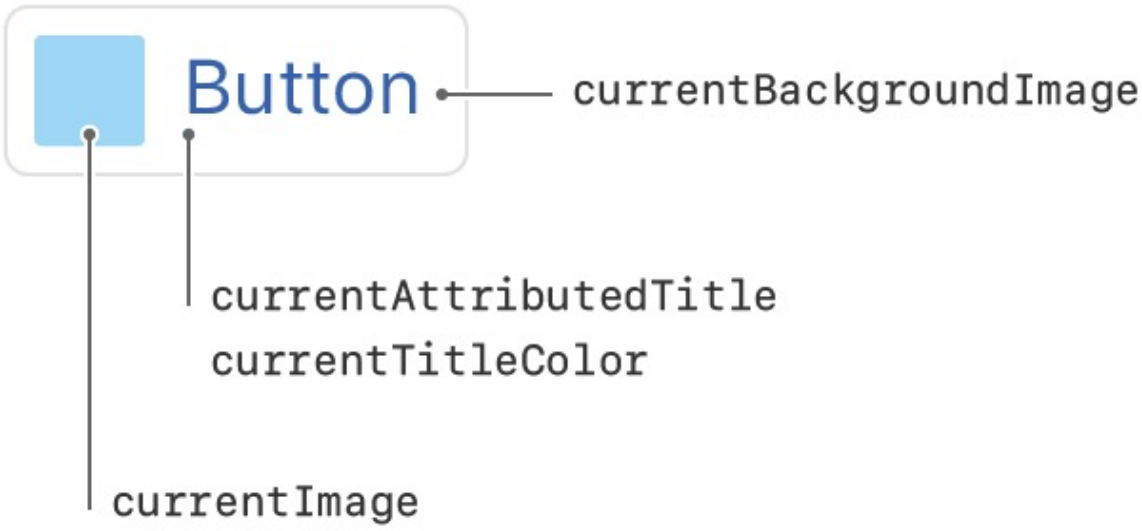
10. adjustFontSizetoFitWidth: UILabel을 설정할 때 글씨의 길이가 설정해 놓은 크기보다 커질 경우 폰트 크기를 줄일 것인지를 결정하는 프로퍼티.
- YES : 설정해 놓은 크기보다 글씨의 길이가 길 경우 폰트 크기를 줄인다.
- NO(기본값) : 설정해 놓은 크기보다 글씨의 길이가 길 경우 '_'으로 대체한다.

UIButton

버튼을 누르거나 포커스가 있는 버튼을 선택하면 버튼에 연결된 작업이 수행된다. 텍스트 레이블, 이미지 또는 둘 다를 사용하여 버튼의 용도를 전달할 수 있다.

버튼에는 기본(default), 강조표시(highlighted), 포커스(focused), 선택(selected), 사용안함(disabled)의 다섯가지 상태가 있다. 인터페이스에 버튼을 추가하면 처음에는 기본 상태가 되고 사용자가 버튼을 사용하면 다른 값으로 변경된다.

버튼의 내용은 문자열또는 이미지로 구성할 수 있고 이를 UILabel과 UIImage 객체로 구성할 수 있다.



버튼의 이미지 및 제목을 모두 지정할 수 있다. 표시된 속성을 사용하여 버튼의 현재 내용에 접근할 수 있다.

UIButton의 프로퍼티 및 메소드

1. init(type:UIButton.ButtonType) : 지정된 유형의 새 버튼 생성하고 반환.

2. backgroundColor : UI의 배경색을 설정하는 프로퍼티. UIColor클래스 형태로 설정한다.

3. setTitle:forState : 버튼의 상태에 해당하는 글씨를 설정하는 메소드. setTitle에는 NSString 클래스를, forState에는 UIControlState를 설정하며 UIControlState의 주요인자는 다음과 같다.
- UIControlStateNormal : 일반적일 때
- UIControlStateHighlighted : 선택 중으로 되었을 때
- UIControlStateDisabled : 비활성화가 되었을 때
- UIControlStateSelected : 선택이 되었을 때

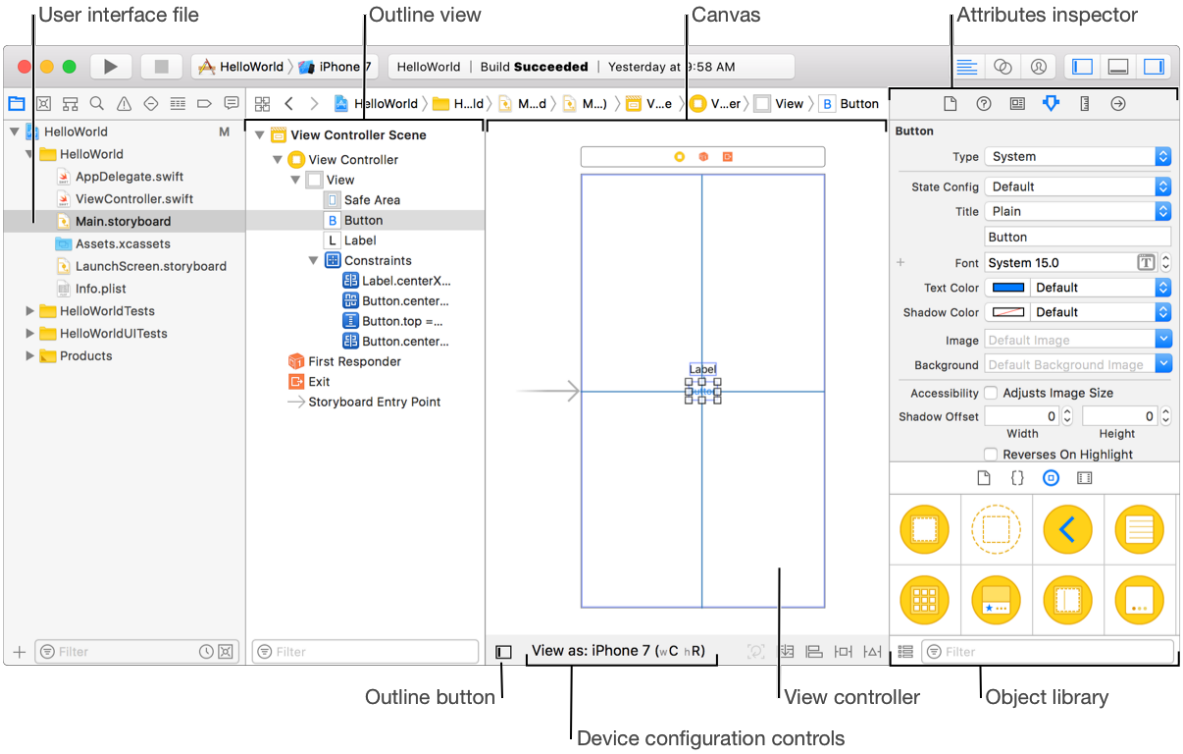
4. setTitleColor:forState : 버튼의 상태에 해당하는 글씨의 색을 설정하는 메소드. setTitleColor에는 UIColor 클래스를, forState에는 UIControlState를 설정

5. setBackgroundImage:forState: 버튼의 상태에 해당하는 배경 그림을 설정하는 메소드. setBackgroundImage에는UIImage 클래스를, forState에는 UIControlState를 설정

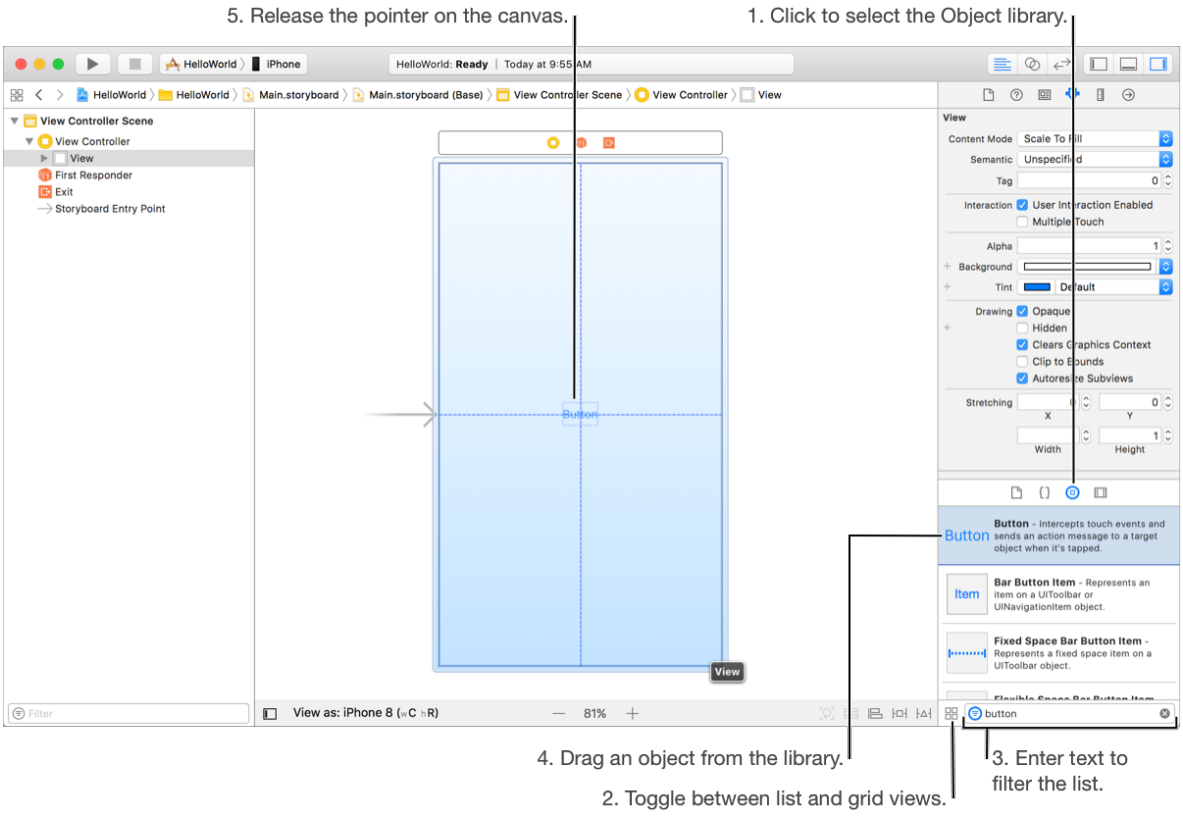
6. addTarget:action:forControlEvents: 버튼의 현재 상태에 따라 호출할 메소드를 설정하는 메소드. addTarget에는 action의 target이 되는 class를, action에는 호출할 메소드를 @selector 형태로, forControlEvents에는 UIControlEvents를 설정. UIControlEvents의 주요인자는 다음과 같다
- UIControlEventTouchDown : Control을 터치했을 때
- UIControlEventTouchDragEnter : Control에 손가락을 밀어 넣었을 때
- UIControlEventTouchUpInside : Control의 안쪽에서 손가락을 땠을 때
- UIControlEventTouchUpOutside : Control의 밖에서 손가락을 땠을 때
- UIControlEventEditingDidBegin : 편집이 시작되었을 때
- UIControlEventEditingChanged : 편집한 데이터가 변경되었을 때
- UIControlEventEditingDidEnd : 편집이 완료되었을 때

7. buttonType : 버튼의 종류를 설정하고 가져오는 프로퍼티. 주요 버튼의 종류는 아래와 같다.
- UIButtonTypeCustom : 사용자 설정 버튼
- UIButtonTypeRoundedRect : 둥근 테두리의 네모 버튼
- UIButtonTypeDetailDisclosure : > 버튼이 앞에 달린 버튼
- UIButtonTypeInfoLight : 인포 버튼이 밝은 형태로 있는 버튼
- UIButtonTypeInfoDark : 인포 버튼이 어두운 형태로 있는 버튼
- UIButtonTypeContactAdd : + 버튼이 앞에 달린 버튼

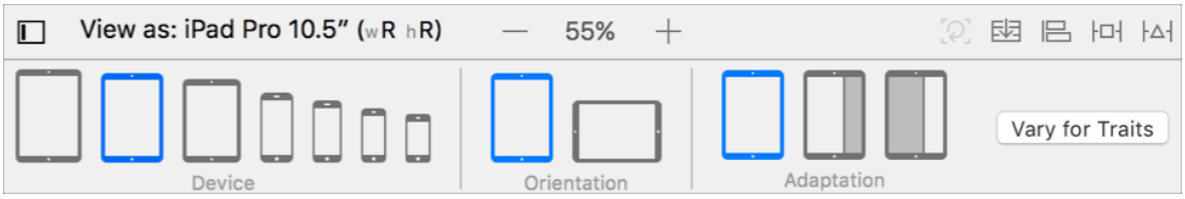
Xcode에서 만들기



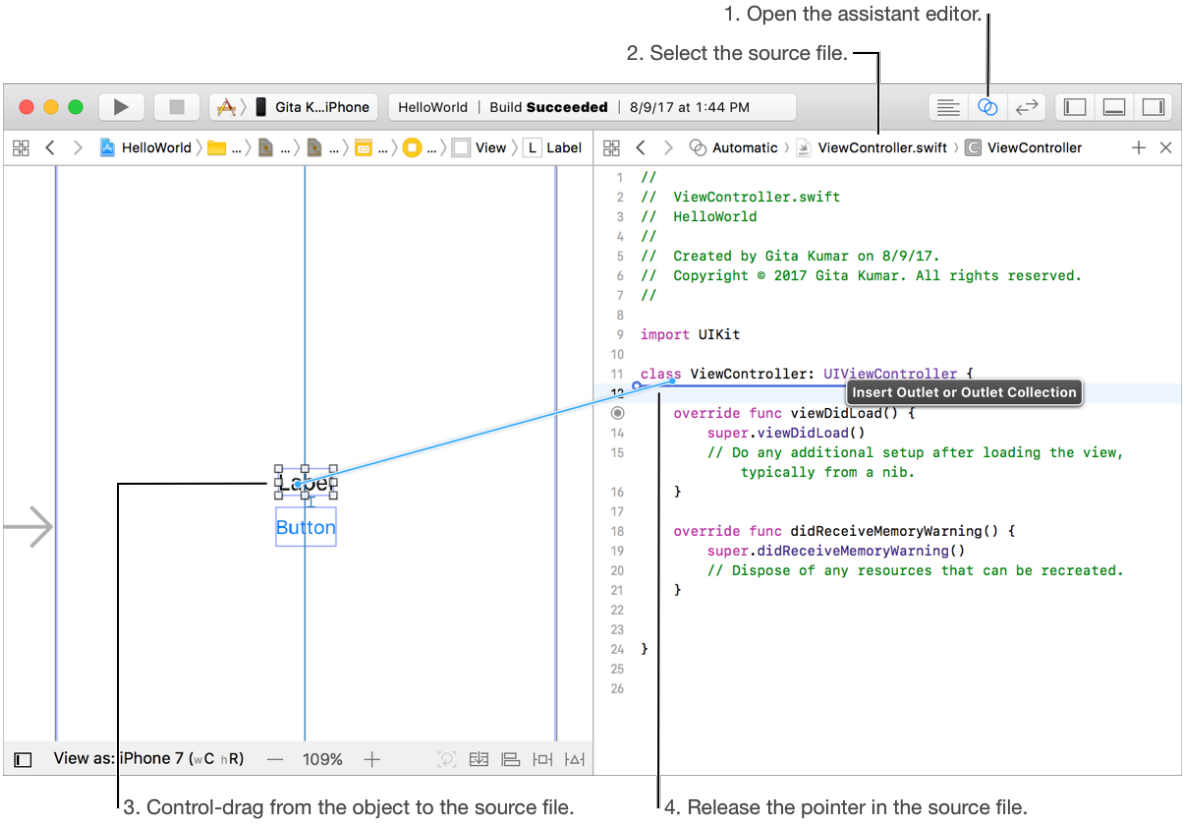
iOS앱의 UI에 화면 요소들을 간단하게 컨트롤 하기 위해서 애플은 Interface Builder라는 것을 제공하고 있는데 그 것이 스토리보드(Storyboard)가 그 역할을 하고있다. UI를 변경하거나 확인하기 위해선 스토리보드 파일을 열어야 하고 기본적으로 프로젝트를 생성하면 자동으로 생기는 Main.storyboard와 LaunchScreen.storyboard 파일이 생긴다.LaunchScreen.storyboard의 경우 앱이 처음 실행될 때(Launching)뜨는 화면으로 그 화면의 UI를 변경하거나 확일할 때 사용한다.



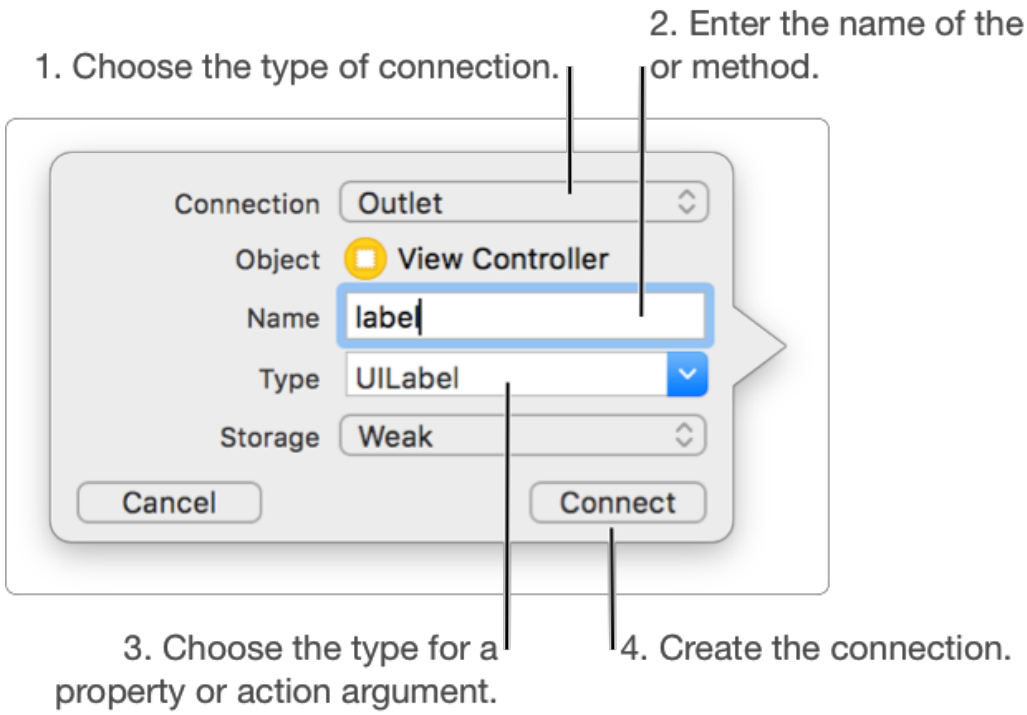
화면에 버튼을 추가하는 방법은 위의 사진을 토대로 Object library에서 버튼을 드래그해 원하는 곳에 배치시킨다. 화면의 방향을 바꾸거나 여러 기기에 대응하는 화면을 확인해보고 싶다면 아래에 [View as: 기기명] 이라고 써져있는 Device configuration controls 부분을 클릭하면 아래와 같은 화면이 나타나기 때문에 다양하게 변경하면서 확인할 수 있다.



코드들과 UI가 현재는 따로 구분되어 있기 때문에 현재는 해당 버튼에 접근할 수 있는 방법이 없다. 이때는 Storyboard의 Interface Builder기능을 활용해야 한다. 아래의 사진처럼 해당 버튼 혹은 텍스트나 다른 화면 요소들을 원하는 Controller클래스가 있는 부분에 컨트롤(Ctrl)을 누른채 드래그 해서 올려놓기만 하면 자동으로 처리가 된다.



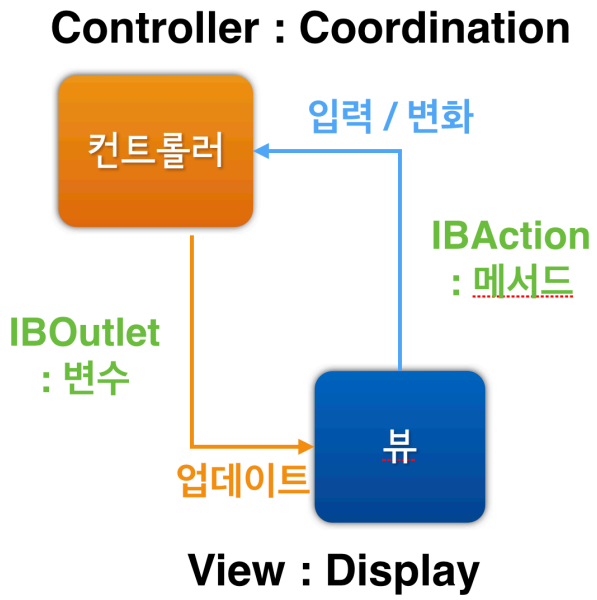
Ctrl + 드래그 를 마친순간 아래와 같은 팝업창이 뜨는데 해당 팝업창에 원하는 Connection 타입과 Name(프로퍼티 이름) 그리고 Type(프로퍼티의 타입)을 정의해줘야 한다.



IBOutlet과 IBAction은 Connection 타입 중 하나로 각각 프로퍼티와 메소드로 생성되어 화면 요소들을 컨트롤 할 수 있는 연결고리가 된다. 이렇게 설정해놓은 경우 내부에 구현되어 있는 코드에 의해 Inteface Builder 기능이 적용되어 자동으로 화면 요소와 소스코드가 매핑되어 간단히 접근이 가능하다.

- IBOutlet과 IBAction은 똑같이 View(화면-Storyboard)와 Controller를 매핑시켜주지만 역할은 다르다

IBOutlet vs. IBAction



위 그림처럼 IBAction의 경우 유저(사용자)를 통한 특정 이벤트를 감지해서 Controller에 알리는 역할을 하고, IBOutlet은 처리 결과를 View단에 알려서 원하는 동작을 이끌어 낸다.