IBOutlet과 IBAction을 사용할 때 주의할 점들

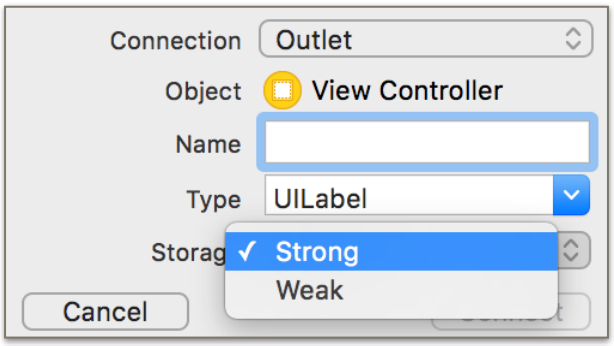
@ <- Annotation : 여러 가지 역할을 하지만 핵심은 주로 변수나 메소드의 성격을 알려주는 역할은 한다. 단, 작업자에게 알려주는 것이 아니라 컴파일러에게 알려주는 역할이다.

@IBOulet은 프로퍼티에, @IBAction은 메소드에 각각 추가되는데 이 어노테이션이 붙어있는 프로퍼티나 메소드는 인터페이스 빌더에 관련된 것이라는 의미를 나타낸다.

@IBOulet, @IBAction를 모아서 인터페이스 빌더 어노테이션이라고 부른다.
이 어노테이션이 붙은 프로퍼티나 메소드는 처음에 앱이 빌드될 때 컴파일러가 체크하고 연결 정보를 찾아 인터페이스 빌더의 객체와 서로 연결해 준다.

@IBOulet - 객체의 참조

@IBOulet은 화면상의 객체를 소스 코드에서 참조하기 위해 사용하는 어노테이션이다.
주로 객체의 속성을 제어할 목적으로 클래스의 프로퍼티에 연결한다.
@IBOulet으로 정의된 프로퍼티를 **아울렛 변수**라고 부른다.



Storage 항목
Strong과 Weak 타입의 차이는 메모리 회수 정책에 있다.
일반적으로 변수나 상수는 다른 곳에서 참조되고 있을 경우 메모리에서 제거되지 않는 것이 원칙이지만, Weak 타입으로 선언된 변수나 상수는 다른 곳에서 참조되고 있더라도 시스템이 임의로 메모리에서 제거할 수 있다.

지정된 값이 지워질 수도 있음에도 불구하고 Weak 타입이 필요한 이유는 메모리 관리의 이슈 때문이다. Strong 타입 객체들끼리 상호 참조되는 일이 발생하는 경우, 어떤 경우에도 참조 카운트가 0이 되지 않으므로 애플리케이션이 실행되는 한 영원히 메모리에서 제거되지 않는다. 이는 메모리 누수로 이어진다. 이때 상호 참조되는 객체의 어느 한쪽을 Weak 타입으로 지정하면 시스템에 의해 임의로 제거가 가능하므로 순환되는 상호 참조로부터 벗어날 수 있다. 이 과정을 충분히 이해하려면 ARC의 구조나 원리에 대한 폭넓은 이해가 필요하다.

@IBAction - 객체의 이벤트 제어

@IBAction은 객체의 이벤트를 제어할 때 사용하는 어노테이션이다.
버튼을 눌렀을 때 화면을 이동시키거나 메시지를 띄워 주는 등, 특정 객체에서 지정된 이벤트가 발생했을 때 우리가 의도하는 일련의 프로세스를 실행케 할 목적을 갖는다.
이를 위해 이 어노테이션은 메소드와 함께 사용되는데, 이를 **액션 메소드**라 부른다.

대부분의 객체와 연결할 수 있는 아울렛 변수와 달리 액션 메소드는 버튼이나 테이블 셀 등 사용자와 상호 반응할 수 있는 객체를 연결할 때만 사용할 수 있다는 제한이 있다.

정리
객체에 대한 속성이나 이벤트를 클래스 파일과 연결할 때 속성은 Outlet으로 연결하고, 이벤트는 Action으로 연결한다. 객체를 클래스 파일에 연결하면 소스 코드에 @ 표시와 함께 @IBOulet이나 @IBAction 어노테이션으 붙으므로 컴파일러는 해당 변수나 메소드가 인터페이스 빌더와 연결된 것임을 파악할 수 있다.

자신이 자주 사용하는 앱의 여러 ViewController에서 라이프사이클 (viewDidLoad, viewWillAppear, viewDidAppear 등등)에 따라 어떤 동작들이 이루어지고 있을지 상상해보기

viewDidLoad() : 이 메소드는 뷰의 로딩이 완료 되었을 때 시스템에 의해 자동으로 호출되기 때문에 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.
viewDidAppear(_:) : 뷰가 완전히 화면에 표현되고 난 다음에 호출된다.

화면이 처음 실행되거나 또는 퇴장한 상태에서 다시 등장하기 시작하는 상태 (Appearing 상태)로 바뀌는 동안 viewWillAppear(_:) 메소드가 호출된다. 화면이 등장할 때마다 데이터를 갱신해 주고 싶다면, 이 메소드를 오버라이드하여 원하는 코드를 작성하면 된다.
화면이 완전히 등장하고 나면 viewDidAppear(_:) 메소드가 호출된다. 이 상태에서 다른 액션이 일어나 화면의 전환이 이루어지거나 홈 버튼을 눌러 앱이 백그라운드로 내려가는 등 스키린에서 화면이 퇴장하는 상태 변화가 발생하면 그 즉시 viewWillDisappear(_:) 메소드가 호출되고, 상태변화가 완료되었을 때 viewDidDisappear(_:) 메소드가 호출된다.

>> 아이폰의 캘린더 App
캘린더 App 클릭 -> viewWillAppear(_:) -> 달력이 화면에 뜸 -> viewDidAppear(_:) -> 1월 29일 클릭 -> 달력 화면의 viewWillDisappear(_:) -> 달력 화면의 viewDidDisappear(_:) -> 특정 날짜 상세 화면의 viewWillAppear(_:) -> 특정 날짜 상세 화면이 뜸 -> 특정 날짜 상세 화면의 viewDidAppear(_:) -> ...

프로그래밍에서 디자인 패턴이란 무엇인가, 디자인 패턴이 가지는 의미는 무엇일까

프로그램 구성에 대한 디자인 즉, 구조적 설계를 의미. 객체지향 프로그래밍에서는 객체 간의 관계가 중요한데, 설계에 관한 문제를 해결하기 위한 해법으로 객체들의 관계를 구조화한 사례가 일반화된 것이 디자인패턴이다.

프로그래밍에서는 다양한 설계 문제를 합리적으로 해결하기 위한 목적으로 디자인 패턴들이 등장했다. 특정 객체의 값이 바뀌었을 때 여러 객체에 이를 알려주기 위한 방법, 구조를 바꾸지 않고 기능을 변경하거나 개선하기 위한 방법, 하나의 객체만 생성하도록 보장하기 위한 방법 등이 이에 해당하는 문제들이라 할 수 있다.

알아두면 좋을 패턴들
- 팩토리 패턴 / 옵저버 패턴 / 데코레이터 패턴 / 싱글톤 패턴 / 어댑터 패턴 / 이터레이터 패턴 / 델리게이트 패턴