Chapter06. Delegate, Notification

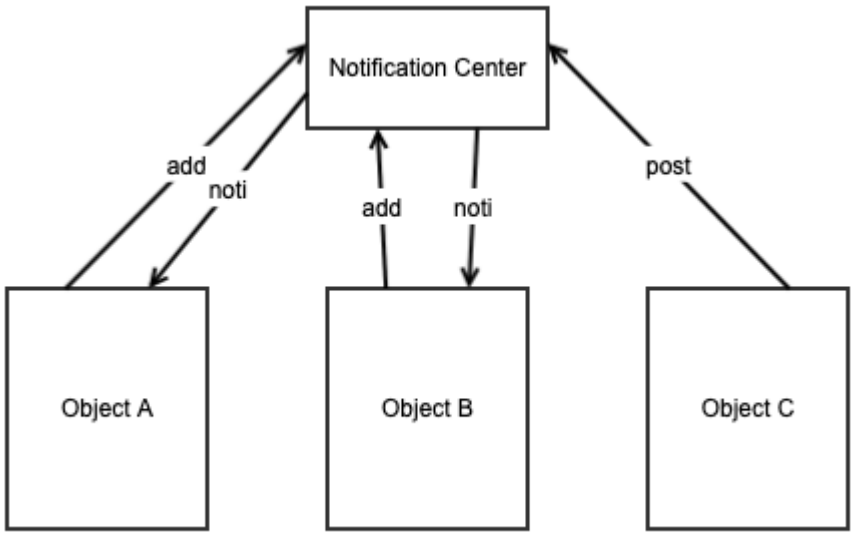
1. 델리게이트 패턴 (Delegate Pattern)

- Delegate : 대리자, 위임자
- 하나의 객체가 해야 하는 일 중 일부를 다른 객체에 넘기는 것
- iOS 앱을 이루는 기본 아키텍처의 대부분은 델리게이트 패턴에 의존

장점
- 매우 엄격한 Syntax로 인해 프로토콜에 필요한 메소드들이 명확하게 명시됨
- 컴파일 시 경고나 에러가 떠서 프로토콜의 구현되지 않은 메소드를 알려줌
- 로직의 흐름을 따라가기 쉬움
- 프로토콜 메소드로 알려주는 것뿐만이 아니라 정보를 받을 수 있음
- 커뮤니케이션 과정을 유지하고 모니터링하는 제 3의 객체(ex: NotificationCenter 같은 외부 객체)가 필요없음
- 프로토콜이 컨트롤러의 범위 안에서 정의됨

단점
- 많은 줄의 코드가 필요
- 많은 객체들에게 이벤트를 알려주는 것이 어려움

2. 노티피케이션 패턴 (Notification Pattern)



Notification Center라는 싱글턴 객체를 통해서 이벤트들의 발생 여부를 옵저버를 등록한 객체들에게 Notification을 post하는 방식으로 사용

장점
- 많은 줄의 코드가 필요없이 쉽게 구현이 가능
- 다수의 객체들에게 동시에 이벤트의 발생을 알려줄 수 있음
- Notification과 관련된 정보를 Any? 타입의 object, [AnyHashable: Any]? 타입의 userInfo로 전달할 수 있음

단점
- key 값으로 Notification의 이름과 userInfo를 서로 맞추기 때문에 컴파일 시 구독이 잘 되고 있는지, 올바르게 userInfo의 value를 받아오는지 체크가 불가능함
- 추적이 쉽지 않을 수 있음
- Notificaiton post 이후 정보를 받을 수 없음

3. KVO (Key Value Observing)

- A 객체에서 B 객체의 프로퍼티가 변화를 감지

장점
- 두 객체 사이의 정보를 맞춰주는 것이 쉬움
- new/old value를 쉽게 얻을 수 있음
- key path로 옵저빙하기 때문에 nested objects도 옵저빙이 가능

단점
- NSObject를 상속받는 객체에서만 사용 가능
- dealloc될 때 옵저버를 지워야함

덧붙여서 Swift 4 전까지는 NSObject의 메소드인 observeValue(forKeyPath:change:context:)를 오버라이드하여 옵저버를 추가했으나 Swift4부터는 구독하고 싶은 프로퍼티에 observe()를 추가하여 클로저로 사용할 수 있게 하였다. 그러나 Swift 상에서는 didSet이나 willSet 같은 것으로 충분히 대체가 가능할 것 같아 굳이 써야하나 싶은 패턴인 것 같다.

참고)
https://medium.com/@Alpaca_iOSStudy/delegation-notification-%EA%B7%B8%EB%A6%AC%EA%B3%A0-kvo-82de909bd29