2주차 멘토링 로그

1. Gamja :
    a. 조건문에 따라 true/false 로 반환하는 경우 조건문을 return을 하는 편이 낫다
2. Brix :
    a. Double? 은 Optional<Double>과 같다
    b. Optional은 컨테이너이기 때문에 map을 메서드로 갖는다
    c. map은 인자로 함수나 클로저를 받는데, 인자로 `numberFormatter.string` 이라는 **함수**를 넣어준 것
    d. 과도하게 코드를 함축하는 것은 좋지 않지만, 의미만 명확히 드러난다면 일률이 향상되면서도 가독성도 좋은 코드이다.
    e. 함수형 프로그래밍은 코드가 길어지면 의미가 불명확해지는 것을 예방할 수 있기 때문에 사용한다
    f. computed property로 사용할 변수는 lazy var로 선언되면, 반복적으로 생성/해제 되지 않는다.

       다만, 클로저 내에서 사용할 때는 조심해야 한다.
3. Blu :
    a. 가독성을 위해 탈출 상황이 존재하는 경우, **guard** 문으로 의미를 명확하게 드러내는 것이 좋다
    b. **defer**는 함수가 종료되기 전에 반드시 실행되야할 필요가 있을 때 사용하면 된다.
    c. enum은 처리할 변수의 범위를 제한할 용도로 사용하면 된다.
    d. 리팩토링은 시간날 때 마다 계속…
    e. 한 메서드가 너무 많은 일을 하는 것은 좋지 않다.
        i. unwindToLogin에서 validation까지 도맡아 한다는 점에서 개선의 여지가 있었다.
4. Sjsj : ...
5. Nois :
    a. 의미를 명확히 할 필요가 있을 때 Scope를 나누면 도움이 됨

b. String += 는 컴파일러가 적절하게 처리하지 못해 컴파일 속도를 굉장히 느려지게 만듦으로, append를 사용하는 것이 좋다.
c. 코드를 작성할 때 핵심인 코드들이 분산되어 있지 않고, 한 곳에 몰려 있도록 작성하면 가독성이 좋아진다.
d. validation의 경우가 enum으로 처리하기 좋은 예이다.
e. flatMap은 Optional 결과를 꺼내어 저장하고, nil일 경우는 버려진다.

질문 답변

1. 스토리보드가 Swift 코드보다 뒤늦게 메모리 할당이 되기 때문이기도 하고, Objective-C 코드에서는 Optional이라는 개념이 존재하지 않기 때문
2. **UITabBarItem**이 **UIBarItem**을 상속해서 만들어진 클래스
3. 해당 기업의 기존 구성원의 성향에 따라 다르지만, 보편적으로는
    a. 대기업 : CS 기초가 충실한 사람
    b. 스타트업 : 포트폴리오가 탄탄한 사람
4. 커스텀 컨트롤 뷰를 코드로 사용하고 싶은 경우, AppDelegate에서 didFinishLaunchingWithOptions 메서드에서
   코드용 초기화 - override init(frame: CGRect)를 이용해 뷰를 생성해서 window?.rootViewController에 대입해 주어서 사용한다
5. Swift Code에서는 사용되지 않고, Storyboard에서만 사용하는 프레임워크인 경우에는 Swift Compiler의 최적화에 의해 제거되어 참조가 되지 않으므로, 이를 막기 위해 명시적으로 Link를 해주어야 한다
   (아론 힐리가스의 iOS 프로그래밍 - 5판, 99페이지)
6. 모든 view에는 rootViewController가 할당이 된다, 하나의 앱이 window를 여러개 쓰는 경우는, 미러링용 앱인 경우가 대표적
7. 대부분의 기능은 loadView가 아닌, viewDidLoad로 해결할 수 있으므로, 왠만해서는 loadView를 호출하지 않는 것이 좋다, Storyboard, Nib를 사용해서 구성되는 View인 경우에는 사용하지 말고, 커스텀 컨트롤인 경우 Code로 View를 띄운다는 것을 보여주고 싶을 때는 상관 없을 것 같음
8. viewWillAppear에 레이아웃 구성 코드를 넣지는 말고, 모델 변경에 대한 View 단에 대한 처리 및 반영을 해주면 된다.

9. IBInspectable을 실제로 많이 사용 함,
10. TDD가 프로젝트에 테스트를 붙인다고 되는 것이 아니라, 설계 단계에서 부터 유닛 테스트를 고려해서 프로젝트를 설계하는 것을 말함,
    a. 테스트를 어떻게 할지를 정확히 이해해야 가능함, 고급 개발자가 되면 가능할 것이라 생각
    b. iOS에서는 현업에서는 실제로 잘 이뤄지기가 힘듦, 주변 개발자를 설득하는 것도 힘들고 마감기한을 맞추기도 더 어려워지기 때문에
    c. 프로젝트가 거대해져서, 개인에 의존해서는 시스템이 유지가 되기 힘들때는 TDD 까지는 아니더라도, Unit Test를 붙여가며 하는 것이 생산성에 더 도움이 된다.
11. Storyboard에서 확실히 지원이 되지 않는 UI에는, 코드의 도움을 받아서 하는 것이 좋다.
12. 네이밍은 한 눈에 이해가 되기만 하면 어떻게 짓든 상관없다…
13. Boundary Condition : 경계 조건
    a. 숫자의 경우 : 0, 음수, Max, Min 등...
14. UI Test는 지금은 쓰지말자…, 미래엔 바뀔수도
15. 예시) 로그인 이후의 테스트
    a. setUp - 로그인
    b. tearDown - 로그아웃
16. 국내 앱들은 거의 적용을 안함… 적용 난이도가 어렵기 때문에
    - 능력과 여유가 생긴다면 꼭 적용해보자
17. 오토레이아웃을 이용해 부모 View의 비율로 정하고 싶다면, width, height에 multiply를…
18. 디자이너의 요구사항 변경이 잦은데, 스냅킷을 이용하면 처음에는 만들기 힘들어도 대응하기는 쉬워짐

- 다음 주 화요일에 피드백을 해주신다고… 멘션 부탁하셨음
- 최근에 iOS에서는 MVVM을 적용하고자 한다면 Redux를