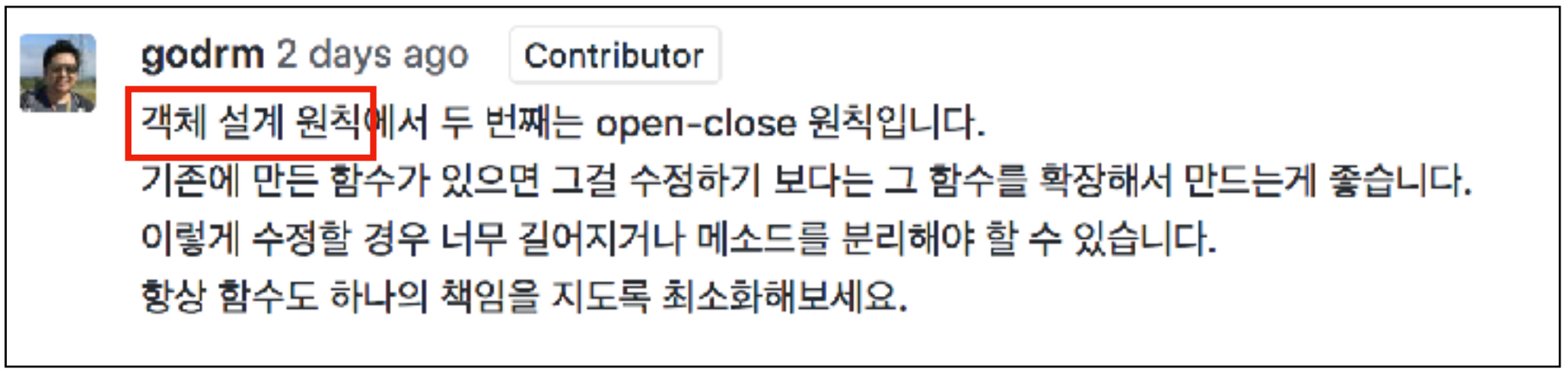
SRP 를 적용해보자

SRP(Single Responsibility Principle) : 단일 책임 원칙

시작은 간단한 피드백



godrm 2 days ago Contributor

객체 설계 원칙에서 두 번째는 open-close 원칙입니다.
기존에 만든 함수가 있으면 그걸 수정하기 보다는 그 함수를 확장해서 만드는게 좋습니다.
이렇게 수정할 경우 너무 길어지거나 메소드를 분리해야 할 수 있습니다.
항상 함수도 하나의 책임을 지도록 최소화해보세요.

- 객체 설계 원칙 이란 ?

두문자	약어	개념
S	SRP	**단일 책임 원칙 (Single responsibility principle)** 한 클래스는 하나의 책임만 가져야 한다.
O	OCP	**개방-폐쇄 원칙 (Open/closed principle)** "소프트웨어 요소는 확장에는 열려 있으나 변경에는 닫혀 있어야 한다."
L	LSP	**리스코프 치환 원칙 (Liskov substitution principle)** "프로그램의 객체는 프로그램의 정확성을 깨뜨리지 않으면서 하위 타입의 인스턴스로 바꿀 수 있어야 한다." 계약에 의한 설계를 참고하라.
I	ISP	**인터페이스 분리 원칙 (Interface segregation principle)** "특정 클라이언트를 위한 인터페이스 여러 개가 범용 인터페이스 하나보다 낫다."[4]
D	DIP	**의존관계 역전 원칙 (Dependency inversion principle)** 프로그래머는 "추상화에 의존해야지, 구체화에 의존하면 안된다."[4] 의존성 주입은 이 원칙을 따르는 방법 중 하나다.

Open Close = 개방 폐쇄

우선 지적받은 개방 폐쇄 규칙 먼저 체크

- 기능을 변경하거나 확장 할수 있으며
- 그 기능을 사용하는 코드는 수정하지 않는다

정리 하자면

- 기능을 만들고 나서 더 많은 기능이 필요하게 되면
- 그 함수를 수정하지 말고 확장이나 변경을 하자

개방 폐쇄 규칙에 어긋난 이유

1. 기능추가를 하고 싶다 !
2. 저기있는 함수를 쓰자 !
3. 이 함수에 이 기능도 넣자 !
4. 규칙 위반 !
5. 단일책임 원칙도 어긋났습니다…. !

단일 책임 원칙 : SRP(Single Responsibility Principle)

= 한 함수는 최소한의 책임만 가지게 한다.

- 지적받은 함수의 책임

1. 사용자의 입력을 받음
2. MyLine, MyPoint 형태인지 확인
3. MyLine 형태가 아닌데 MyPoint 결과값이 2 이상이 아닌지 확인
4. 좌표값의 범위 확인
5. 이상의 작업중 문제가 생기면 처음부터 다시 반복

나쁜 단일책임의 예

- 사람을 객체로 만든다고 가정
- 남여의 구분을 변수로 구분
- 남자는 군대를 가야하기 때문에 군번 변수 추가

```
class 사람 {
    let 성별 : 성별
    let 군번 : String
}
```

- 이 경우 여자.군번 사용을 막을수 없음

좋은 단일책임의 예

- 사람을 객체로 생성
- 남여가 따로 사람객체를 상속
- 남자만의 변수인 군번추가
- 혹은 사람객체에 군번함수를 넣어 상속된 객체의 속성별로 생성

단일 책임 원칙 : SRP(Single Responsibility Principle)

= 한 함수는 최소한의 책임만 가지게 한다.

- 지적받은 함수의 책임

    1. 사용자의 입력을 받음
    2. MyLine, MyPoint 형태인지 확인
    3. MyLine 형태가 아닌데 MyPoint 결과값이 2 이상이 아닌지 확인
    4. 좌표값의 범위 확인
    5. 이상의 작업중 문제가 생기면 처음부터 다시 반복

길다

하지만
필요한 기능들

```swift
/// 정규식을 통과할때까지 유저입력을 받는 함수
func receiveUserPoint()->Array<MyPoint>? {
    // 유저입력값을 저장하기 위한 변수
    var userPoint = ""
    // 결과값을 저장하기 위한 배열 선언
    var myPointList : Array<MyPoint>?

    // 정규식을 이용하기 위해서 커터 선언
    let extracter = Extracter()
    //정규식을 통과할때까지 반복
    repeat {
        // 입력을 위한 안내 메세지 출력
        print("좌표를 입력하세요. 예시: (12,4)")
        // 유저입력값을 저장
        userPoint = receiveUserInput()
        // 1개 좌표단위로 자름
        guard let PointList =
            extracter.extractPointFrom(userPoint: userPoint)
            else {
            continue
        }
        guard let myPointListCheck =
            extracter.makeMyPointListFrom(confirmedPointList:
            PointList) else {
            continue
        }
        myPointList = myPointListCheck
        if !extracter.isitLineIn(userPoint: userPoint) &&
            myPointList!.count > 1 {
            print("잘못된 좌표입니다.")
            myPointList = nil
        }
    } while myPointList == nil
    return myPointList
}
```

정답은

쪼개기
&
메인함수

```swift
func main(){
    // 인풋뷰 구조체 선언
    let inputView = InputView()
    // 체크 기능을 이용하기 위해 체커 선언
    let checker = Checker()
    // 제대로 된 값을 입력할때 까지 입력을 반복
    var repeatFlag = true
    // 유저입력값을 반복문 밖으로 전해줄 변수
    var userPoints = ""
    // 검증을 통과 못하면 입력부터 다시 받는다
    repeat {
        // 사용자 입력을 받는다
        let userInput = inputView.receiveUserInput()
        // 입력한 형태가 좌표형태가 맞는지 체크
        guard checker.isCorrectPointType(latters: userInput) else {
            continue
        }
        // 좌표값이 범위 내인지 체크
        guard checker.checkPointRange(latters: userInput) else {
            continue
        }
        // 검증을 통과하면 반복을 중지한다
        repeatFlag = false
        // 검증이 끝난 입력값을 밖으로 내보냄
        userPoints = userInput
    } while repeatFlag

    // 통과한 좌표값을 정규식화 한다
    let regexedPoints = Extracter.extractPointFrom(originLatters: userPoints)!
```

이후의 과제

- MyPoint 와 MyLine 을 어떻게 만들고 보낼것인가

  이전 방법 : MyPoint 배열을 만들어서 전달

  - 아웃뷰 에서 해당 배열을 받아서 포문으로 출력

  안좋은 방법이라고 지적받음

- JK 의 피드백 : 객체를 리턴하자
- 린생의 힌트 - 팩토리 패턴
- 프로토콜을 사용하는 패턴으로 해결방법으로 보여짐