# Week 1-1



## 기초가 뭔데?

여러분이 생각하는 코드와 프로그래밍의 기초는 무엇인가요?

코드의 기초에 대해 알아봅시다.

### Swift API Programming Guide [Link]

- 한꺼번에 모두 적용하려고 하면 힘듭니다
- 한 번에 하나씩, 점진적으로 늘려갑니다

## 객체지향 프로그래밍

## SOLID

- SRP(Single-Responsibility Principle, 단일책임의 원칙)
  - 한 클래스는 단 한가지의 **책임**만을 가져야한다
  - **책임**은 **변경의 이유**다
    - 변경을 위해 수정이 되려면 많은 내용이 수정 → 응집도가 높음
    - 변경을 위한 이유가 같은 것들끼리 모이자
    - 수정은 한곳에 집중되야함 → 여러 곳에 걸친 수정이 안이루어짐 → 결합도 낮음
- OCP(Open-Close-Principle, 개방-폐쇄의 원칙)
  - **확장에 열려**있고 **변경에 닫혀**있다
  - 확장을 할때는 기존의 코드를 최대한 건드리지않고 확장하자
  - 만약 기존의 코드를 수정하게되면 연쇄적인 수정을 하지 않을 수 있게 하자
  - 기존 코드의 수정은 버그가능성이 있고, 그것을 테스트 해야한다
- LSP(Liskov Substitution Principle, 리스코프 치환 원칙)
  - 자식클래스는 부모 클래스로써의 역할을 완벽히 할 수 있어야한다
- ISP(Interface Segregation Principle, 인터페이스 분리의 원칙)
  - 클라이언트가 불필요하게 자신이 사용하지 않는 인터페이스에 의존하지 말아야한다
    - 불필요한 인터페이스를 의존하면 상속받은 메소드를 퇴화시켜야하는 경우가 발생할 수 있음
    - 불필요한 인터페이스에 의존하여 불필요한 빌드가 유발될 수 있음
    - 큰 인터페이스를 작은 인터페이스들로 분리하고, 필요한 부분만 클라이언트가 취사선택하여 사용할 수 있게 해야함 → P.O.P.
- DIP(Dependency Inversion Principle, 의존성 역전의 원칙)
  - 상위 수준의 모듈은 하위수준의 모듈에 의존해서는 안된다
  - 구체적인 사항은 추상화에 의존해야한다
    - 구체적인 것은 잘 변한다
    - 추상적 인건 잘 변하지 않는다

## 객체지향 생활체조

1. 한 메서드에 오직 한 단계의 들여쓰기만 합니다
2. else 표현을 사용하지 않습니다
3. 모든 원시 값과 문자열을 포장합니다

4. 한 줄에 점을 하나만 사용합니다
5. 이름을 줄여 쓰지 않습니다(축약 금지).
6. 모든 엔티티를 작게 유지합니다
7. 3개 이상의 스위프트 기본 데이터타입(Int, String, Double 등) 프로퍼티를 가진 타입을 구현하지 않습니다
8. 일급 콜렉션을 사용합니다
9. getter/setter를 구현하지 않습니다

## 디미터 법칙

- 디미터 법칙은 모듈은 자신이 조작하는 인스턴스의 속사정을 몰라야 한다는 법칙입니다
- 인스턴스는 자료를 숨기고 메서드를 공개합니다. 즉, 인스턴스는 조회 메서드로 내부 구조를 공개하면 안됩니다.
- 다음 코드는 디미터 법칙을 어기는 것으로 볼 수 있습니다.

```
let output: String = text.options().scratch().absolutePath()
```

## 타입의 설계

### 타입은 작아야 합니다

- 타입을 만들 때 첫 번째 규칙은 크기입니다. 타입은 작아야 합니다.
  두 번째 규칙도 크기입니다. 더 작아야 합니다.
- SOLID의 단일 책임 원칙(Single Responsibility Principle, SRP)은 타입이나 모듈을 변경할 이유가 하나, 단 하나뿐이어야 한다는 원칙입니다.
- 타입은 책임, 즉 변경할 이유가 하나여야 한다는 의미입니다.
- 응집도(cohesion) - 타입은 인스턴스 프로퍼티 수가 적어야 합니다.
- 응집도를 유지하면 작은 타입 여럿이 나옵니다.
- 큰 메서드를 작은 메서드 여럿으로 쪼개다 보면 종종 작은 타입 여럿으로 쪼갤 기회가 생깁니다. 그러면서 프로그램에 체계가 더 잡히고 구조가 더 투명해집니다.

### 변경하기 쉬운 타입

- 요구사항은 변화기 마련입니다. 따라서 코드도 변하기 마련이죠.

- 구현 타입에 의존하게 되면 테스트가 어려우며, 변화에 빠르게 대응하기 어렵습니다. 변화에 따르게 대응하려면 DIP 원칙을 지키는 습관을 가져야 합니다.
- SOLID의 DIP(Dependency Inversion Principle) 원칙은 타입이 상세한 구현이 아니라 추상화(프로토콜 등)에 의존해야 한다는 원칙입니다.
- 테스트가 가능할 정도로 시스템 결합도를 낮추면 유연성과 재사용성도 더 높아집니다.

### 예외 타입

- 인스턴스는 추상화 뒤로 자료를 숨긴 채 자료를 다루는 메서드만 공개합니다
- 자료 구조는 자료를 그대로 공개하며 별다른 메서드는 제공하지 않습니다
    - 자료 구조체의 전형적인 형태는 공개 프로퍼티만 있고 메서드가 없는 인스턴스입니다. 이런 자료 구조체를 Data Transfer Object(DTO)라고 합니다.

# 아하! 모먼트

## 라이브러리, 프레임워크

- Library
- Framework

## 디자인패턴, 아키텍쳐

- Design Pattern
- Architecture

# 이력서 피드백 I

## Step 0

1. 가고 싶은 회사의 방향성을 정합니다
    - 교육이면 교육, 메신저면 메신저, GIS면 GIS, 헬스케어면 헬스케어, 메타버스면 메타버스 등
    - 글에 방향이 있으면 메시지가 더욱 전달이 잘 됩니다
        - 이는 보는 사람으로 하여금 더 만나보고 싶게 해줍니다
    - 방향이 다양하다면 그만큼의 이력서를 준비할 수 있도록 합니다
        - 특히 클래스의 상속을 연상하며 super-이력서와 sub-이력서를 구분해놓으면 좋습니다
2. 일관된 어투를 사용하도록 합니다
3. 디자인을 하는 것도 좋으나, 규격은 있어야 합니다
    - 폰트의 크기와 색깔 등을 어떤 부분에 어떻게 사용할지 정하고 들어갑니다
        - 타이틀 폰트 24pt, 타이틀 색상 빨강 등
    - 색은 아름답게 쓰되 복잡하지 않아야 합니다
        - 흔히 3가지 색상만 조합하라고 조언합니다
    - 디자인이든 색깔이든, 어렵다면 구글링을 통해 하나의 탬플릿을 찾아 사용하도록 합니다
        - 탬플릿을 사용한다면 되도록 색깔이나 디자인을 더 추가하지 않습니다. 특히 색깔
        - 덜어내는 건 괜찮습니다
4. 국내회사에 지원한다면, 한글을 사용해서 작성합니다
    - 필요하고 어필하고 싶다면, 따로 영문 이력서를 준비하고 국문과 영문 이력서를 모두 전달합니다
5. 링크는 되도록 빼는 게 좋습니다
    - 인사담당자가 개발팀에게 종이로 갖다주면 링크를 클릭할 방법이 없어요
        - 인쇄를 고려한다면 링크는 QR코드로 대체해도 괜찮습니다
        - 꼭 필요한 링크는 축약 URL 등을 사용하여 텍스트 링크로 적어주세요
    - 또 이력서와 포트폴리오를 검토하는 사람은 정말 바쁩니다
    - 보통 포폴을 읽는 사람은, 이게 중요하지 않아서 링크로 대체했다고 생각합니다
    - 그래서 일단 중요한 건 문서 안에 다 담겨있어야 합니다
    - (중요) 진짜로 부수적인 건 링크를 달되, 어떤 부분을 보라고 가이드를 해주는 게 좋습니다

6. 경력자를 뽑는 것에 쫄지 않습니다
    - 밑져야 본전입니다. 일단 넣는 게 좋습니다
    - 또 꼭 경력을 안받아도 되는데, 회사 사정상 공고를 경력직으로 내는 경우도 왕왕 있습니다
    - 안되면 안되는가보다 하고 넘어가면 됩니다
7. 문서는 되도록 PDF 문서로 전달합니다
    - PDF 문서는 표준 포멧으로 플랫폼에 종속적이지 않아, 받는 입장에서 읽고 관리하기 좋아요

## 이력서

- 면접을 위한 수단입니다
- 면접에 가서 질문을 받고 싶은 내용들로 채워져야 합니다
- 자신은 없지만 써놓으면, 면접에서 질문이 들어올 가능성이 매우 큽니다
    - 그 때 답변을 못하면 오히려 마이너스 요소로 작용합니다
- 거짓말이나 과장을 해놓으면 면접에서 들키니까 안 그러는 게 좋고, 그럴 필요도 없습니다