# 2장, 3장

**2장 : 의미 있는 이름**

- 의도를 분명히 밝혀라
  - 변수, 함수, 클래스 등에 의도가 드러나도록 이름을 지어라
  - 의도가 드러나면 코드의 이해와 변경이 쉬워진다.
    - ex) 코드는 동일하지만 변수명만 바꿔 명확해진 코드 예시

- 그릇된 정보를 피하라
  - 다르게 해석될 수 있는 이름을 변수명으로 하지마라
    - ex) userlist(리스트 구조 아님)
  - 비슷한 이름 여러개를 쓰지마라
  - 유사한 개념은 유사한 표기법을 사용하지만(정보), 일관성이 떨어지면 그릇된 정보이다.
  - ex) 0, O, 1, l, I `(X)`

- 의미 있게 구분하라
  - 읽는 사람이 차이를 알도록,
  - 이름이 정보를 제공하도록 이름을 지어야 한다. (의미 없게 짓지 말아라)
    - ex) a1, a2, a3 / moneyamount, money `(X)`
    - ex) source, destination `(O)`

- 발음하기 쉬운 이름을 사용하라
  - 프로그래밍은 사회 활동이기에 발음하기 쉬운 이름도 중요하다.

- 검색하기 쉬운 이름을 사용하라
  - 문자 하나를 사용하든지 상수를 그냥 사용하면 검색하기 어렵다.

    - 여러 곳에서 사용한다면 검색하기 쉬운 이름이 적합하다.
        - 이름 길이는 범위 크기에 비례해야 한다.
        - ex) 5 : 상수, 검색 어려움 vs WORK_DATS_PER_WEEK : 검색 쉬움

- 인코딩을 피하라
    - 인코딩 = 구닥다리, 읽기 어려움
    - 써야 하더라도 알기 쉽도록 써야한다.

- 자신의 기억력을 자랑하지 마라
    - i, j, k같이 전통적인 것을 제외하고
    - 자신만이 이해하는, 기억하는 변수로 이름 짓지말아라, 타인이 이해하는 이름 지어라.

- 클래스 이름
    - 명사, 명사구가 적합하다.
    - ex) customer, account `(O)`
    - ex) data, info `(X)`

- 메서드 이름
    - 동사, 동사구가 적합하다.
    - ex) payment, save `(O)`
    - 접두사 get, set, is

- 기발한 이름은 피하라
    - 재치있는 이름보다 명료한 이름이 좋다.
    - 특정 문화를 모르면 이해하기 어렵다.

- 한 개념에 한 단어를 사용하라

    - 추상적인 개념에 한 단어만 사용해라
        - ex) controller = manager = driver 다 같음

- 말장난을 하지 마라
    - 같은 맥락에만 같은 단어를 사용해라
    - ex) add ⇒ 더해서, 이어서 / 반환하기 `(O)`
    - ex) add ⇒ 집합에 추가하기 `(X)`, insert가 옳음

- 해법 영역에서 가져온 이름을 사용하라
    - 코드 어차피 프로그래머가 읽는다.
    - 전산, 알고리즘, 패턴, 수학 용어 등 사용해도 된다. (프로그래머 용어)

- 문제 영역에서 가져온 이름을 사용하라
    - 문제 영역과 관련 깊다면 문제 영역에서 이름을 지어라
    - 나중에 코드 보수하는 프로그래머가 분야 전문가에게 의미 물어서 파악할 것이다.

- 의미 있는 맥락을 추가하라
    - 변수 스스로 의미가 분명하지 않다면
    - 맥락을 추가하여 의미를 구분하여 이해하기 쉽도록 한다.

- 불필요한 맥락을 없애라
    - 의미가 분명한 경우 짧은 이름이 더 좋다.
    - 불필요하게 맥락을 추가하지 마라

- 마침
    - 우리는 모든 이름을 암기하지 못하기에 암기는 도구에 맡기고
    - 우리는 코드가 읽히기 쉽도록 짜야한다.

**3장 : 함수**

- 작게 만들어라!
  - 함수는 작게 만들어야 좋다.
  - 블록과 들여쓰기
    - if/else, while문 등에 들어가는 블록은 한 줄이어야 한다.
    - enclosing function도 작아지고 블록 안에서 호출하는 함수 이름도 적절히 지으면, 코드 이해도 쉬워진다.
    - 중첩 구조가 1, 2단을 넘어서지 않도록 해야 한다는 것이다.(들여쓰기)

- 한 가지만 해라!
  - 함수는 한 가지를 해야 한다. 그 한 가지를 잘 해야 한다. 그 한 가지만을 해야 한다.
  - 한 가지 : 추상화 수준이 하나인 단계

- 함수 당 추상화 수준은 하나로!
  - 함수에서 추상화 수준이 동일해야 한다. (섞으면 안된다.)
  - 위에서 아래로 코드 읽기 : 내려가기 규칙
    - 코드는 위에서 아래로 이야기처럼 읽혀야 한다.
    - 위에서 아래로 추상화 수준이 낮아져야 한다.

- Switch 문
  - 추상 팩토리를 통해 switch문을 숨긴다.

- 서술적인 이름을 사용하라!
  - 함수를 읽으면서 짐작했던 기능을 그대로 수행한다면 좋은 코드이다.
  - 이름이 길어도 좋다. 일관성이 있어야 한다.

- 함수 인수
  - 함수 인수는 적을수록 좋다. (없는 것이 가장 좋다)

        - 적을수록 읽는 사람이 이해하기 쉽다.
    - 단항 인수
        - 질문을 던지는 경우
        - 변환해 결과를 반환하는 경우
        - 이벤트 함수인 경우
        - 위 3가지를 제외하고 가급적 피한다.
    - 플래그 인수
        - 함수가 한꺼번에 여러 가지를 처리한다는 것
        - 추함
    - 이항 함수
        - 이해하기 어려움
        - 실수 위험
    - 삼항 함수
        - 더 이해하기 어려움
    - 인수 객체
        - 인수가 2-3개면 독자적인 클래스로 변환하여 선언하라.
    - 인수 목록
        - 가변인수 : list형 인수 하나로 취급
    - 동사와 키워드
        - 함수의 의도나 인수의 순서와 의도를 제대로 표현하기 위해 좋은 이름이 필요하다.
        - 동사 + 키워드 형태로 적는다.

- 부수 효과를 일으키지 마라!
    - 한 가지만 하도록 해야 한다.
    - 다른 작업이 꼭 필요한 경우 함수 이름에 분명히 명시해야 한다.
    - 출력 인수
        - 쓰면 안된다.

        - 객체 지향 언어에서 this를 쓴다.
        - 함수에서 상태를 변경해야 한다면 함수가 속한 객체 상태를 변경하는 방식을 택한다.

- 명령과 조회를 분리하라!
    - 함수는 뭔가를 수행하거나 뭔가에 답하거나 둘 중 하나만 해야한다.
        - 객체 상태 변경 or 객체 정보 반환
    - 명령과 조회를 분리하여 혼란을 뿌리뽑는다.

- 오류 코드보다 예외를 사용하라!
    - try/catch 블록 뽑아내기
        - try/catch 블록은 별도 함수로 뽑아내라
        - 정상동작과 오류처리 동작을 분리하면 코드를 이해하고 수정하기 쉬워진다.
    - 오류 처리도 한 가지 작업이다.
        - 오류 처리도 한 가지 작업이므로 오류를 처리하는 함수도 오류만 처리해야 한다.
        - try/catch 깔끔하게 써라
    - 오류코드 의존성 자석
        - 오류코드가 변한다면 다 재컴파일 재배치를 해야하므로 번거롭다.
        - 오류코드 대신 예외를 사용해라

- 반복하지 마라!
    - 중복되면 나중에 손 봐야 할 곳이 중복된다는 것이다.
    - 중복이 없으면 가독성도 높아진다.

- 구조적 프로그래밍
    - 함수를 작게 만들면 단일 입출구 규친 안지켜도 된다.
    - goto는 쓰지마라

- 함수를 어떻게 짜죠?
  - 글짓기 처럼 짜라
  - 초안은 규칙도 어기고 지저분하지만,
  - 다듬어 가면서 완성한다.

- 결론
  - 함수는 동사고 클래스는 명사다.
  - 시스템은 풀어나아갈 이야기이다.
  - 함수가 분명해야 이야기를 풀어나가기 쉬워진다.