Clean Code

새롭게 알게 된 것들

발표자:우연서

읽게 된 이유

✅ 유명한 책

✅ 시간 제한이 걸린 과제를 하면서 잘 읽히는 코드가 필요

✅ 더 나은 코드에 대한 논리적인 설명이 필요

공유할 내용

❌ 전체적인 책 내용 요약

✅ 그럼에도 중요하다고 생각하는 것

✅ 이유가 인상적이엿던 것

✅ 리뷰 받았던 경험

네이밍

이름을 붙이기

1. 의도를 명확히 하기
2. 그릇된 정보를 피하자
3. 의미있게 구분하자
4. 발음하기 쉬운 이름을 사용하자
5. 검색하기 쉬운 이름을 사용하자
6. 인코딩을 피하자
7. 자신만 기억하는 이름을 짓지 말자
8. 클래스이름은 명사구로 시작한다
9. 메서드 이름은 동사, 동사구로 시작한다
10. 기발한 이름은 피하자
11. 한 개념에 한 단어만 사용하자
12. 한 단어를 두가지 목적으로 사용하지 말자
13. 기술 용어로 표현할 수 있다면 사용하자
14. 기술 용어로 표현 할 수 없다면 개발하는 문제 영역의 지식을 이름으로 결정하자
15. 의미 있는 맥락을 추가하자
16. 불필요한 맥락은 굳이 쓰지 않는다

의미있게 구분하자

❌ product === productImfo === productData

→ 같은 의미라면 뒤에 쓸모 없는 단어를 붙이지 않는다.

❌ theZork => zork

→ 불필요한 접두어도 붙이지 않는다.

❌ NameString => Name

→ 타입을 변수에 포함시키지 않는다.

❌ textVariable => text

→ 이미 변수인데 또 변수인 것을 나타내지 않는다

인코딩을 피하자

타입이나 인터페이스의 특징을 변수에 담지 말기

❌ NameString => Name

→ 타입을 변수에 포함시키지 않는다.

❌ IShapeFactoryImp => ShapeFactoryImp 또는 CShapeFactory

→ 인터페이스 정보를 굳이 담아야 하겠다면 Imp 를 붙이자

한 개념에는 하나의 단어만 사용하기
(내가 자주 실수하는 부분)

❌ fetch get retrieve 등 어려 단어로 사용하면 안된다

→ 처음에 정한 단어로 통일한다.

한 단어를 두가지 목적으로 사용하지 않기

❌ `add, insert` 는 다른 역할이다.

→ add(x,y)=>x+y, insert(arr, x)=>[…arr, x] **다른 역할.** addPage라고 하면서

기술 용어를 사용해서 이름을 짓는다

👍🏻 JobQueue

👍🏻 batteryAdapter

→ 일 목록을 담는 JobQueue인 것도 알고

→ 베터리 변환의 역할을 하는 어뎁터 패턴을 사용한 것도 안다.

(기술용어가 없을 때) 문제 영역에서 이름 짓는다

→ 개발하는 분야의 영역에서 이름을 따오는 것

불필요한 이름은 지우자

→ 개발하는 분야의 영역에서 이름을 따오는 것

함수

함수를 작게 나눠서 작성하자

1. 작게 만들기
2. **한가지 역할**만 해야한다
3. Switch 문 사용 주의
4. 서술적인 이름 사용
5. **함수의 인수**는 짧게!
6. **사이드 이펙트**!!!
7. 명령과 조회의 분리
8. 오류코드로 분기처리하기 보다는 예외로 처리하자
9. 반복되는 걸 함수로!
10. 구조적 프로그래밍 : break, continue, goto는 자제

한가지 역할

① 추상화 수준이 같은 상태

```
Layer1 : PAGE render
설정 페이지 포함, 테스트 페이지 포함, 해제 페이지 포함

Layer2 : detail render
설정페이지 함수 내부
테스트 페이지 내부
하단 페이지 구현
```

JavaScript

② 의미 있는 이름으로 다른 함수를 추출 할 수 있다면 여러 작업을 하는 상태

🧐 왜 추상화 수준을 하나로 유지해야 할까?
- 읽는사람이 어려움. 근본 개념인지 세부사항인지 구분이 어려움
- 세부사항을 계속 한 함수에 추가해서 함수가 길어 질 수 있다

함수의 인수

🧐 왜 함수의 인수를 줄여야 할까

① 읽는 사람이 이해가 어렵게 됨

② 테스트가 어려워짐

③ 함수와 인수가 추상화 수준이 다르게 될 확률이 높아진다.

함수의 인수

🧐 왜 함수의 인수를 줄여야 할까

① 읽는 사람이 이해가 어렵게 됨

② 테스트가 어려워짐

③ 함수와 인수가 추상화 수준이 다르게 될 확률이 높아진다.

🧐 출력인수는 입력인수보다 이해하기 어렵다

```
append(report) // report가 붙는다는건지, 어디에 붙는다는 건지 알 수 없음
report.append(Html html) // 인수도 없애고 report에 붙인다는 것을 알 수 있음
```

JavaScript

출력인수는 무엇일까? : 인수가 다시 출력이 되는 인수..???

사이드 이펙트

함수 안에서 외부함수를 조작하면 생기는 것

함수 외부가 영향이 받도록 하지 말기

```
function checkPassword(String name , String password){
  if(isUser(name) && isPassword(password){
    // do something
  }else {
    Session.initialize() // 심각!!! 이거 안됨
  }

}
```

JavaScript

객체와 자료구조

객체와 자료구조

- 왜 객체와 자료구조를 사용하는지
- 객체와 자료구조의 차이점이 무엇인지
- 코드에서 두 형식에 접근하는 방식의 차이가 있는지

객체와 자료구조

뭐가 다를까

자료구조

```javascript
class Point {
 constructor(x, y) {
   this.x = x;
   this.y = y;
 }
}

let point = new Point();
```

- 값에 바로 접근
- DTO 같은것이 자료구조체

객체

```javascript
interface Point {
  getX();
  getY();
  setCartesian(x,y);
  getR();
  getTheta();
  setPolar(r,theta);
}
```

- 객체는 접근 방법이 고정되어 있음.
- 값을 숨기고 함수를 보여주는 것

객체와 자료구조

🧐 그럼 함수로만 값을 감추면 다 객체인가?

놉!

추상화 과정이 들어가야 함.

```
class Vehicle {
  getFuelTankCapacityGallons(); // 남은 겔론 크기
  getGallonsOfGasoline(); // 총 몇 겔론 용량이 남았는지
}
```

JavaScript

Better

```
class Vehicle {
  getPercentFuelRemaining()
}
```

JavaScript

객체와 자료구조의 사용

자료구조를 활용한 절차지향

```typescript
class Square {
  topLeft: Point;
  side: number;
  constructor(left: Point, side: number) {
    this.topLeft = left;
    this.side = side;
  }
}

class Rectangle {...}

class Circle {...}

class Geomety {
  PI: number = 3.141592;
  area(shape: Circle | Rectangle | Square) {
    if (shape instanceof Circle) {
      return shape.radious * shape.radious;
    }else if(shape instanceof Rectangle){

    }...
  }
}
```

TypeScript

절차 지향보다 객체 지향이 더 나은가? ⇒ **놉.**

- 절차지향은 새로운 함수를 추가하기 더 쉽다. 새로운 자료구조를 추가하기는 어려움
- 객체지향은 새로운 자료구조를 추가하기가 쉽다. 새로운 함수를 추가하려면 모든 클래스에 다 추가해야 함

객체를 활용한 객체지향

```typescript
interface Shape {
  area(): number;
}
interface Point {
  x: number;
  y: number;
}

class Square implements Shape {
  topLeft: Point;
  side: number;
  constructor(left: Point, side: number) {
    this.topLeft = left;
    this.side = side;
  }
  area(): number {
    return this.side * this.side;
  }
}

class Circle implements Shape {
  radius: number;
  PI: number = 3.141592;
  constructor(left: number) {
    this.number = number;
  }
  area(): number {
    return this.number * this.number * this.PI;
  }
}
```

TypeScript

디미터의 법칙 위반

내부의 객체에 바로 접근을 숨기고, 함수를 공개하기

1. 클래스 C의 메서드
2. f가 생성한 객체의 메서드
3. f 인수로 넘어온 객체의 메서드
4. C 인스턴스 변수에 저장된 객체의 메서드

```
class C {
    private A a;

    private int getArea() { return 0; }

    public void f(B b) {
        getArea(); // 1번의 경우 : Class C의 메서드

        let smalla = new A();
        smalla.setActive(); // 2번 : f 에서 생성한 객체의 메서드

        b.invert(); // 3번 : 인수로 넘어온 객체의 메서드

        a.call() // 4번: 인스턴스 변수에 저장된 객체의 메서드

        Test test = new Test();
        test.print(); // 4번의 경우
    }
}
```

TypeScript

객체를 반환하고 또 반환한 객체가 또 함수를 부르는 형태는 디미터의 법칙을 위반한 것

```
getActiveStatus().goDestination()
```

하지만 객체가 아니라 자료구조의 의미로 접근한건 더럽긴 해도 디미터의 법칙을 위반한 것은 아님

```
getActiveStatus().goDestination()
```

적용해봤는지?

과제에 적용해보자
코드리뷰

- 이름규칙을 먼저 확인
  - 비슷한 역할을 하는데 다른 이름 잡아냄 (내실수)
- Typescript에서 interface명을 ~Imp로 통일
- 함수

```
const mergeItems = (items: Room[]) =>
  items.length ? [...state.items, ...items] : [...state.items];
const nextItems = mergeItems(items);
```

```
items.length ? state.items.concat(items) : [...state.items];
```