

프로그래밍 역량 강화 전문기관, 민코딩

클린코드-1부



목차

1. Naming Rule
2. Method

Naming Rule

스토리 : 호텔 남은 기간

배열 각 칸은, 호텔 방의 남은 사용 일수를 뜻한다.

- x는 0이다.
- 하루가 지날 때 마다 1씩 깎이고,
x는 사용기간이 끝난 일을 나타낸다.

x	x	4	2	x	x	1
---	---	---	---	---	---	---

하루가
지나면

x	x	3	1	x	x	x
---	---	---	---	---	---	---

[도전] 호텔 - 이름을 잘 짓자.

Naming이 지켜진 것일까?

- 적절한 Naming을 지어보자. 의도를 분명하게 나타낸다.
- 아래와 같이 구현 후, Refactor 기능을 사용하여 이름을 변경한다.
 - <https://gist.github.com/jeonghwan-seo/cd12d4b0e4283a52266c04d19685dfe1>

```
void run(vector<int>& theVector) {  
    for (int t = 0; t < theVector.size(); t++){  
        if (theVector[t] == 0) continue;  
        theVector[t] = theVector[t] - 1;  
    }  
}
```

피해야 할 변수명

흔한 약어

- 읽는 사람마다 해석이 다를 수 있다.
- ex) ax, aix, sco, hp 등

대문자 I와 소문자 l, 대문자 O, 소문자 o

- ex) llllllllllll 0000o o O0 o oO0

만능 용어를 자제하자.

어떤 곳이든 다 붙일 수 있는 만능 용어

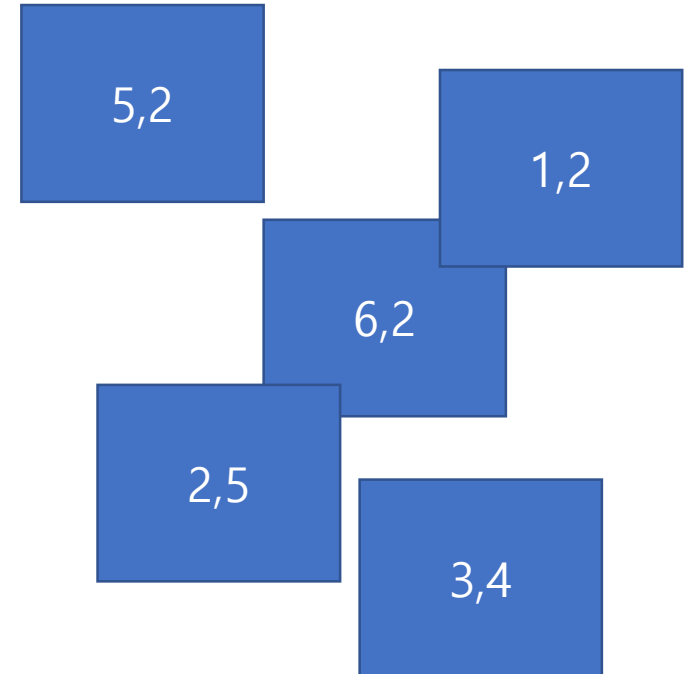
Info, Data, Value

Processor, Manager

이 용어 대신 다른 용어로 변환할 수 있는지 생각해보자.

스토리 : Draw Machine

점을 찍을 좌표를 배열에 저장해두고,
Draw 버튼을 누르면,
점을 찍어주는 프로그램



[도전] Draw Machine Naming 변경

이름을 수정해보자.

- <https://gist.github.com/mincoding-jh/591b25d05238643ea407b36c27f2d74c>

함수명은 동사가 좋다.

- behavior 이기 때문

```
struct xyInfo {  
    int x;  
    int y;  
};  
  
class xyManager {  
public :  
    void add(xyInfo data) {  
        arr.push_back(data);  
    }  
  
    void drawButton() {  
        for (xyInfo o : arr) {  
            cout << o.x << " " << o.y << endl;  
        }  
    }  
  
private:  
    vector<xyInfo> arr;  
};
```

의미있는 이름을 위한 가이드

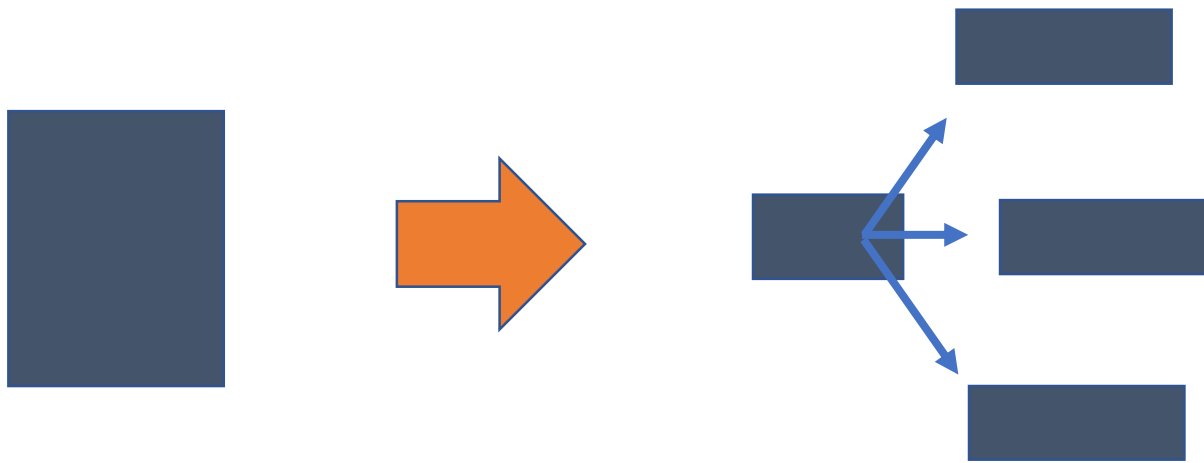
1. 알고리즘이나 패턴을 사용하는 경우, 이를 알 수 있는 이름을 짓는다.
1. Controller, Manager, Driver 등 용어를 명확한 의미를 정하여 사용한다.
1. 발음하기 어려운 이름을 피하자.
1. 읽는 사람이 차이를 알 수 있는 이름을 쓰자.
 - 혼란스러운 예시들
 - moneyAmount vs money
 - theMessage vs Message
 - Customer vs CustomerInfo
 - nameString vs name
 - listGroup vs groupList

Method

함수를 작게 만든다.

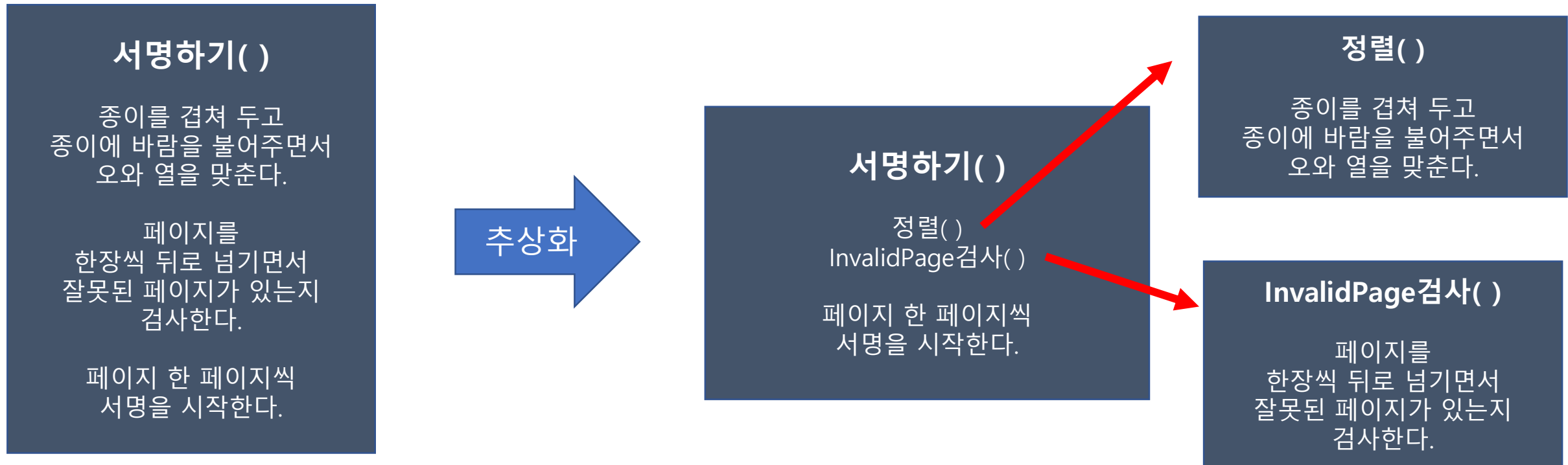
객관적 근거를 대기 어렵지만, 경험적으로
함수 내용이 적을수록 이해하기 좋다.

- 동작이 많은 경우는, 함수를 분리한다.
- 복잡한 내용은 함수로 뺀다.



추상화

코드를 이해하는데 중요하지 않는 부분은
함수로 빼서 추상화하자.



추상화 Level 맞추기

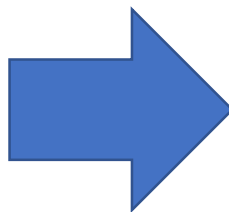
일부만 추상화를 하는 것이 아니다.

추상화를 한다면, 모든 코드가 다 같은 수준(Level)로 추상화를 한다.

서명하기()

정렬()
InvalidPage검사()

페이지 한 페이지씩
서명을 시작한다.



서명하기()

정렬()
InvalidPage검사()

서명()

[도전] 추상화 하기

거대하지 않는, 가독성 있는 makeSign() 메서드 만들기

<https://gist.github.com/mincoding-jh/0baab894d61a47eb435b2079ec31fb2a>

```
class Sign {
public:
    void makeSign(vector<Node>& signVector) {
        //1. 서명 정렬하기
        for (int y = 0; y < signVector.size(); y++) {
            for (int x = y + 1; x < signVector.size(); x++) {
                if (signVector[y].dateCode > signVector[x].dateCode) {
                    Node temp(signVector[y].dateCode, signVector[y].name);
                    signVector[y].dateCode = signVector[x].dateCode;
                    signVector[y].name = signVector[x].name;
                    signVector[x].dateCode = temp.dateCode;
                    signVector[x].name = temp.name;
                }
            }
        }

        //2. valid 검사
        bool flag = false;
        for (Node tar : signVector) {
            if (tar.dateCode > 0 && tar.dateCode < 10) continue;
            flag = true;
            break;
        }

        if (flag == true) {
            throw "Invalid dateCode";
        }
        else {
            //3. 서명하기
            for (Node tar : signVector) {
                cout << tar.dateCode << " : " << tar.name << endl;
            }
        }
    }
};
```

함수는 한 가지 목적을 가져야 한다.

✓ 로버트 C 마틴은, "함수는 한 가지만 해야한다." 라고 말을 한다.

- CPU가 한 가지 Action만을 수행하도록, 함수를 작성하라는 의미가 아니다.
- 너무 작게 만들어서 동작상 의미도 갖지 않는 함수를 만들라고 하는 것이 아니다.

✓ 의미상으로, 한 가지 목적을 수행하는 코드가 되도록 추상화 하자는 의미이다.

- 함수는 의미를 갖는 최소한의 함수로 만든다.

함수 배치 순서

✓ 이해하기 더 편한 소스코드는?

- 위에서 아래로 코드를 읽을 수 있도록 함수를 배치하자.

```
class FuncOrder {  
public :  
    void process2() {  
        //...  
    }  
  
    void process1() {  
        midProcess();  
    }  
  
    void midProcess() {  
        //...  
    }  
  
    void process3() {  
        //...  
    }  
  
    void start() {  
        process1();  
        process2();  
        process3();  
    }  
};
```

```
class FuncOrder {  
public :  
    void start() {  
        process1();  
        process2();  
        process3();  
    }  
  
    void process1() {  
        midProcess();  
    }  
  
    void midProcess() {  
        //...  
    }  
  
    void process2() {  
        //...  
    }  
  
    void process3() {  
        //...  
    }  
};
```

함수 이름 잘 짓기

- ✓ 서술적인 이름을 사용하자.
 - 이름이 길어도 된다. (팀마다 다를 순 있다)
- ✓ 긴 이름은 다음 항목보다는 더 좋다.
 - 짧고 의미를 알 수 없는 이름
 - 서술적인 주석보다 좋다.
- ✓ 예시
 - test □ isTestable
 - insertPart □ includesSetupAndTeardownPages

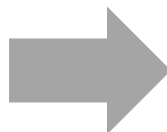
함수 인수 개수

- ✓ 함수 인수는 최소한으로 줄이는 것이, 코드를 이해하기 좋다.
 - `includeSetupPageInto(new PageContent)` vs `includeSetupPage()`
 - 함수 인수 0 개는 가장 이상적이다.
 - 함수 인수 1 개도 이해하기 쉽다. ex) `render(pageData)`

함수 인수에 flag

✓ 함수 안에서 두 가지 동작을 구현해야하는 flag 인자는 피하자.

```
void render(boolean flag) {  
    if (flag) {  
        arrangeText();  
        visualDOMObject();  
    }  
    else {  
        unarrangeText();  
        portingObject();  
    }  
}
```



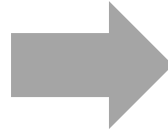
```
void enableRender() {  
    arrangeText();  
    visualDOMObject();  
}  
  
void disableRender() {  
    unarrangeText();  
    portingObject();  
}
```

함수 인수를 더 줄이도록 노력하자.

✓ 함수 인수 2개부터 이해하는데 시간이 더 걸린다.

- writeField(name) vs writeField(outputStream, name)
- 인수 객체를 쓰자
 - makeCircle(int x, int y, double radius)
 - makeCircle(Point axis, double radius)

```
void makeCirlce(int x, int y, double radius) {  
    //...  
}
```



```
class Point {  
    int x, y;  
}  
  
void makeCirlce(Point axis, double radius) {  
    //...  
}
```

결론. 소프트웨어를 작성하는 것은 글짓기와 같다.

- ✓ 글짓기 할 때는, 초안을 작성하고 읽기 좋게 다듬는다.
 - 초안은 대개 서투르고, 어수선하다.
 - 읽는 사람이 내 의도를 정확히 파악할 지 생각해보며, 반복적으로 읽고 글을 고친다.
- ✓ 코딩하는 것은, 글짓기와 같다.
 - 코드를 편하게 작성한다.
 - 그리고 고치는 작업을 시작한다.
 - 코드를 다듬고, 추상화하고, 이름을 바꾸며, 중복을 제거한다.
 - 메서드 순서도 바꾸고 클래스를 쪼개거나 합치곤 한다.
 - 유닛 테스트는 항상 통과되면서 지속적으로 코드를 고친다
- 처음부터 Clean Code가 작성되길 기대하지 말자.
하지만 최종적으로 수정하다 보면 Clean Code가 완성된다.

[도전] 클린 코드로 변경하기

읽기 좋은 코드로 만들기

- <https://gist.github.com/mincoding-jh/18ef2f7bd35e79146c24e01abb4557d0>

```
void drawCircle(bool isDraw, bool isOutline, int x, int y, double radius) {  
    if (x == 0 && y == 0) {  
        firstDraw(radius);  
    }  
    else {  
        if (isDraw) {  
            if (isOutline) {  
                circle_draw2(x, y, radius);  
            }  
            else {  
                circle_draw(x, y, radius);  
            }  
        } else {  
            circleDelete(x, y, radius);  
        }  
    }  
}
```