

읽기 좋은 코드가 좋은 코드다 4,5장

레퍼런스 OS팀

I N D E X

- 01 4장 ~미학~
- 02 5장 ~주식~

01

4장 ~미학~



01

읽기 편한 소스 코드를 작성하기 위한 3가지 원리

코드를 읽는 사람이
이미 친숙한, 일관성
있는 레이아웃 사용

+

비슷한 코드는 서로
비슷해 보이게

+

서로 연관된 코드는
하나의 블록으로

4장 ~미학~

```
1 class StatsKeeper{
2     public:
3         // 일련의 더블 변수값을 저장하는 클래스
4         void add(double d); //그리고 그런 값들에 대한 간단한 정보
5         private int count; //지금까지 몇 개가 저장되었는가
6     public :
7         double Average ();
8
9         private: double minimu;
10        list<double>
11            past_items
12            ; double maxium;
13 };
14
```

보기 안 좋은 코드

- 들어쓰기가 일관적이지 않음
- Public, private: 이 중복으로 쓰임

4장 ~미학~

```
1 // 일련의 더블 변수값을 저장하는 클래스
2 // 그리고 그런 값들에 대한 간단한 정보
3 class StatsKepper{
4 public:
5     void add(double d);
6     double Average;
7
8 public :
9     list <double> past_items;
10    int count; //지금까지 몇 개가 저장되었는가
11
12    double minimu;
13    double maxium;
14 };
```

보기 좋은 코드

- 들여쓰기가
읽기 쉽게 일정함
- **Public,**
Private: 가
한번씩만 쓰임

4장 ~미학~

```
public class PerformanceTester {  
    public static final TcpConnectionSimulator wifi = new TcpConnectionSimulator(  
        500, /* Kbps */  
        80, /* millisecs latency */  
        200, /* jitter */  
        1 /* packet loss % */);  
  
    public static final TcpConnectionSimulator t3_fiber =  
        new TcpConnectionSimulator(  
            45000, /* Kbps */  
            10, /* millisecs latency */  
            0, /* jitter */  
            0 /* packet loss % */);  
  
    public static final TcpConnectionSimulator cell = new TcpConnectionSimulator(  
        100, /* Kbps */  
        400, /* millisecs latency */  
        250, /* jitter */  
        5 /* packet loss % */);  
}
```

코드의 일관성과 간결성의 중요성

- 일관성 X

간결성 X

4장 ~미학~

```
public class PerformanceTester {
    public static final TcpConnectionSimulator wifi =
        new TcpConnectionSimulator(
            500,    /* Kbps */
            80,    /* millisecs latency */
            200,    /* jitter */
            1      /* packet loss % */);

    public static final TcpConnectionSimulator t3_fiber =
        new TcpConnectionSimulator(
            45000, /* Kbps */
            10,    /* millisecs latency */
            0,     /* jitter */
            0      /* packet loss % */);

    public static final TcpConnectionSimulator cell =
        new TcpConnectionSimulator(
            100,    /* Kbps */
            400,    /* millisecs latency */
            250,    /* jitter */
            5       /* packet loss % */);
}
```

코드의 일관성과 간결성의 중요성

- 일관성 O

간결성 X

4장 ~미학~

ONLINE SERVICE GUIDE

```
public class PerformanceTester {  
    // TcpConnectionSimulator(throughput, latency, jitter, packet_loss)  
    //                        [Kbps]    [ms]    [ms]    [percent]  
  
    public static final TcpConnectionSimulator wifi =  
        new TcpConnectionSimulator(500,    80,    200,    1);  
  
    public static final TcpConnectionSimulator t3_fiber =  
        new TcpConnectionSimulator(45000, 10,    0,    0);  
  
    public static final TcpConnectionSimulator cell =  
        new TcpConnectionSimulator(100,    400,    250,    5);  
}
```

코드의 일관성과 간결성의 중요성

- 일관성 ○

간결성 ○

4장 ~미학~

```
DatabaseConnection database_connection;  
string error;  
assert(ExpandFullName(database_connection, "Doug Adams", &error)  
    == "Mr. Douglas Adams");  
assert(error == "");  
assert(ExpandFullName(database_connection, " Jake Brown ", &error)  
    == "Mr. Jacob Brown III");  
assert(error == "");  
assert(ExpandFullName(database_connection, "No Such Guy", &error) == "");  
assert(error == "no match found");  
assert(ExpandFullName(database_connection, "John", &error) == "");  
assert(error == "more than one result");
```

메소드 활용의 중요성

- 메소드 활용 X
- 단점 :
중복된 코드가
많아서 보기 힘들

4장 ~미학~

```
void CheckFullName(string partial_name,
                  string expected_full_name,
                  string expected_error) {
    // database_connection is now a class member
    string error;
    string full_name = ExpandFullName(database_connection, partial_name, &error);
    assert(error == expected_error);
    assert(full_name == expected_full_name);
}
```

메소드 활용의 중요성

- 단점을 보완하기 위한 메소드 제작

4장 ~미학~

```
CheckFullName("Doug Adams", "Mr. Douglas Adams", "");  
CheckFullName(" Jake Brown ", "Mr. Jake Brown III", "");  
CheckFullName("No Such Guy", "", "no match found");  
CheckFullName("John", "", "more than one result");
```

메소드 활용의 중요성

- 메소드를 활용한 코드 간결화

01

미학적 개선의 장점

중복된 코드를
없애서 코드를 더
간결하게 한다.

+

이름이나 에러
문자열 같은
테스트의 중요
부분들이 한 눈에
보이게 모아졌다.

+

새로운 테스트
추가가 훨씬
쉬워졌다.

4장 ~미학~

```
CheckFullName("Doug Adams" , "Mr. Douglas Adams" , "");  
CheckFullName(" Jake Brown ", "Mr. Jake Brown III", "");  
CheckFullName("No Such Guy" , "" , "no match found");  
CheckFullName("John" , "" , "more than one result");
```

```
# Extract POST parameters to local variables  
details = request.POST.get('details')  
location = request.POST.get('location')  
phone = request.POST.get('phone')
```

코드의 열을 맞추면 좋은 점

- 파라미터 인수
파악이 쉬움
- 버그가 눈에 바로
들어옴

4장 ~미학~

```
details = request.POST.get('details')
location = request.POST.get('location')
phone = request.POST.get('phone')
email = request.POST.get('email')
url = request.POST.get('url')
```

메소드 변수 나열은 일관성 있게 중요성

- 변수 - HTML 속 input 필드 순서대로
- 가장 중요한 것 → 가장 덜 중요한 것
- 알파벳 순서

4장 ~미학~

```
class FrontendServer {
public:
    FrontendServer();
    void ViewProfile(HttpRequest* request);
    void OpenDatabase(string location, string user);
    void SaveProfile(HttpRequest* request);
    string ExtractQueryParam(HttpRequest* request, string param);
    void ReplyOK(HttpRequest* request, string html);
    void FindFriends(HttpRequest* request);
    void ReplyNotFound(HttpRequest* request, string error);
    void CloseDatabase(string location);
    ~FrontendServer();
};
```

비슷한 것은 비슷한 것끼리

- 정리 안됨 예시

4장 ~미학~

```
class FrontendServer {
public:
    FrontendServer();
    ~FrontendServer();

    // Handlers
    void ViewProfile(HttpRequest* request);
    void SaveProfile(HttpRequest* request);
    void FindFriends(HttpRequest* request);

    // Request/Reply Utilities
    string ExtractQueryParam(HttpRequest* request, string param);
    void ReplyOK(HttpRequest* request, string html);
    void ReplyNotFound(HttpRequest* request, string error);

    // Database Helpers
    void OpenDatabase(string location, string user);
    void CloseDatabase(string location);
};
```

비슷한 것은 비슷한 것끼리

- 비슷한 것끼리
블록을 만들어서
묶음

4장 ~미학~

```
# Import the user's email contacts, and match them to users in our system.
# Then display a list of those users that he/she isn't already friends with.
def suggest_new_friends(user, email_password):
    friends = user.friends()
    friend_emails = set(f.email for f in friends)
    contacts = import_contacts(user.email, email_password)
    contact_emails = set(c.email for c in contacts)
    non_friend_emails = contact_emails - friend_emails
    suggested_friends = User.objects.select(email__in=non_friend_emails)
    display['user'] = user
    display['friends'] = friends
    display['suggested_friends'] = suggested_friends
    return render("suggested_friends.html", display)
```

**문단은 나누는 게
보기 좋다**

- 코딩에 안 좋은 예

4장 ~미학~

```
def suggest_new_friends(user, email_password):  
    # Get the user's friends' email addresses.  
    friends = user.friends()  
    friend_emails = set(f.email for f in friends)  
  
    # Import all email addresses from this user's email account.  
    contacts = import_contacts(user.email, email_password)  
    contact_emails = set(c.email for c in contacts)  
  
    # Find matching users that they aren't already friends with.  
    non_friend_emails = contact_emails - friend_emails  
    suggested_friends = User.objects.select(email__in=non_friend_emails)  
  
    # Display these lists on the page.  
    display['user'] = user  
    display['friends'] = friends  
    display['suggested_friends'] = suggested_friends  
  
    return render("suggested_friends.html", display)
```

문단은 나누는 게
보기 좋다

- 코딩에 좋은 예

01

5장 ~주식~



5장 ~주식~

```
# remove everything after the second '*'  
name = '*'.join(line.split('*')[:2])
```

설명하지 말아야 할 것

- 코드에서 빠르게
유추할 수 있는
내용은 주식으로
달지 말라

5장 ~주식~

```
1 // 주어진 이름과 깊이를 이용해서 서프트리[h1]에 있는 노드를 찾는다.
2 Node* FindNodeInSubtree(Node* subtree, string name, int depth);
3
4 // 주어진 'name'으로 노드를 찾거나, 아니면 NULL을 반환한다.
5 // 만약 depth <= 0이면 'subtree'만 검색된다.
6 // 만약 depth == N이면 N레벨과 그 아래만 검색된다.
7 Node* FindNodeInSubtree(Node* subtree, string name, int depth);
8
```

**설명 자체를 위한
설명을 달지 말라**

- **달고 싶으면
세부사항을 적어라**

5장 ~주석~

```
1 // 해당 키를 위한 핸들을 놓아준다. 이 함수는 실제 레지스트리를 수정하지 않는다.  
2 void DeleteRegistry(RegistryKey* key);  
3  
4 void ReleaseRegistryHandle(RegistryKey* key);
```

**나쁜 이름에
주석을 달지 마라**

- **대신 이름을
고쳐라**

5장 ~주식~

```
1 // 놀랍게도, 이 데이터에서 이진트리는 해시테이블보다 40퍼 정도 빠르다
2 // 해시를 계산하는 비용이 좌/우 비교를 능가한다.
3
4 // 이 클래스는 점점 꼬이고 있음. 어쩌면 'ResourceNode' 하위 클래스를
5 // 만들어서 정리해야 할 듯.
```

생각을 기록하라

- “감독의 설명”을 포함하라

5장 ~주식~

ONLINE SERVICE GUIDE

```
1 NUM_THREADS = 8 ;  
2  
3 // 이 상수값이 ~~~~ 이다.  
4
```

생각을 기록하라

- 상수에 대한 설명 기록

5장 ~주식~

```
struct Recorder {  
    vector<float> data;  
    ...  
    void Clear() {  
        vector<float>().swap(data); // Huh? Why not just data.clear()?  
    }  
};
```

**코드를 읽는 사람의
입장이 되어라**

- **나올 것 같은 질문
예측하기**

5장 ~주식~

```
// 외부 서비스를 호출하여 이메일 서비스를 호출 (근데 1분 후 에는 타임아웃됨)  
void SendEmail(string to, string subject, string body);
```

코드를 읽는 사람의
입장이 되어라

- 사람들이 쉽게
빠질 것 같은
함정을 경고하라

01

글(주식)을 쓰는 두려움을 떨쳐내라

마음에 떠오르는
생각을 무조건
적어본다.

+

주석을 읽고
무엇이
개선되어야
하는지(그런
부분이 있다면)
확인한다.

+

개선한다

감사합니다.
