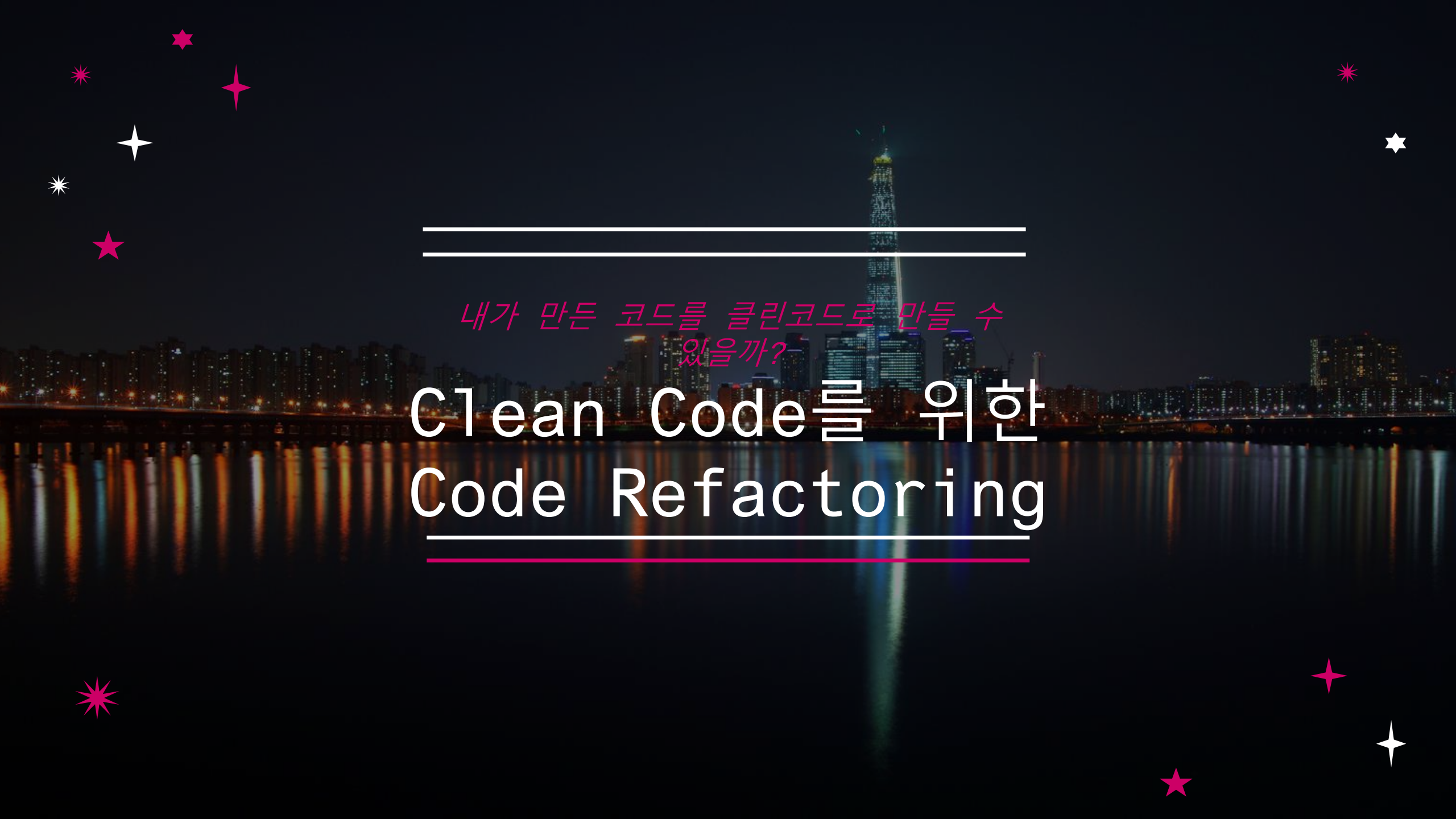
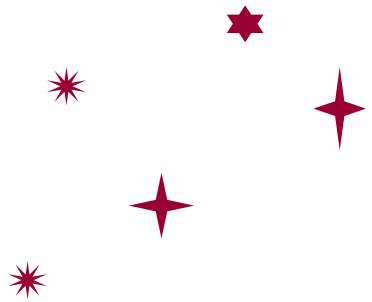


내가 만든 코드를 클린코드로 만들 수 있을까?

Clean Code를 위한 Code Refactoring

CONTENTS



//Clean Code란 무엇인가

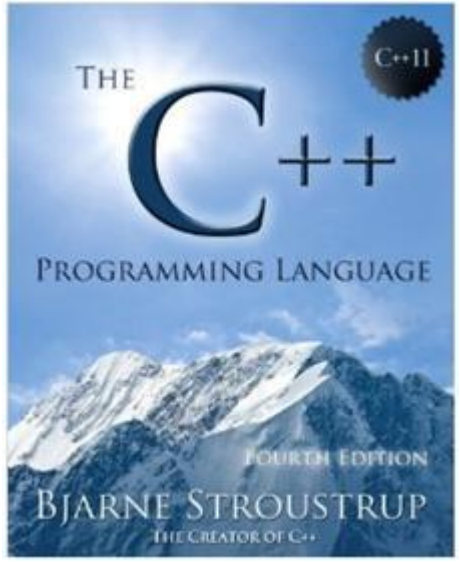
“

I like my code to be **elegant** and **efficient.**

The logic should be **straightforward** and make it hard for bugs to hide,
the dependencies minimal to ease maintenance,
error handling complete according to an articulated strategy,
and performance close to optimal so as not to tempt people
to make the code messy with unprincipled optimizations.

Clean code **does one thing** well.

”



Bjarne Stroustrup, inventor of C++:

“

Clean code is simple and direct.

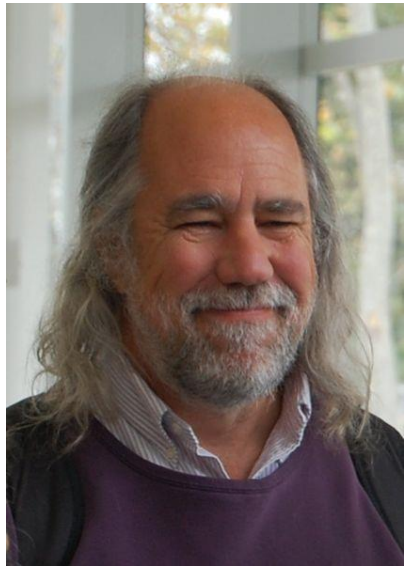
Clean code reads like well-written prose.

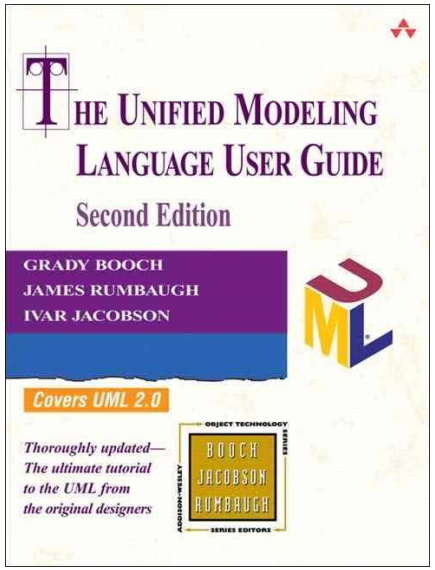
Clean code never obscures the designers’ intent

but rather is full of crisp abstractions and straightforward lines of control.

”

Grady Booch,
author of Object-Oriented Analysis
and Design with Applications





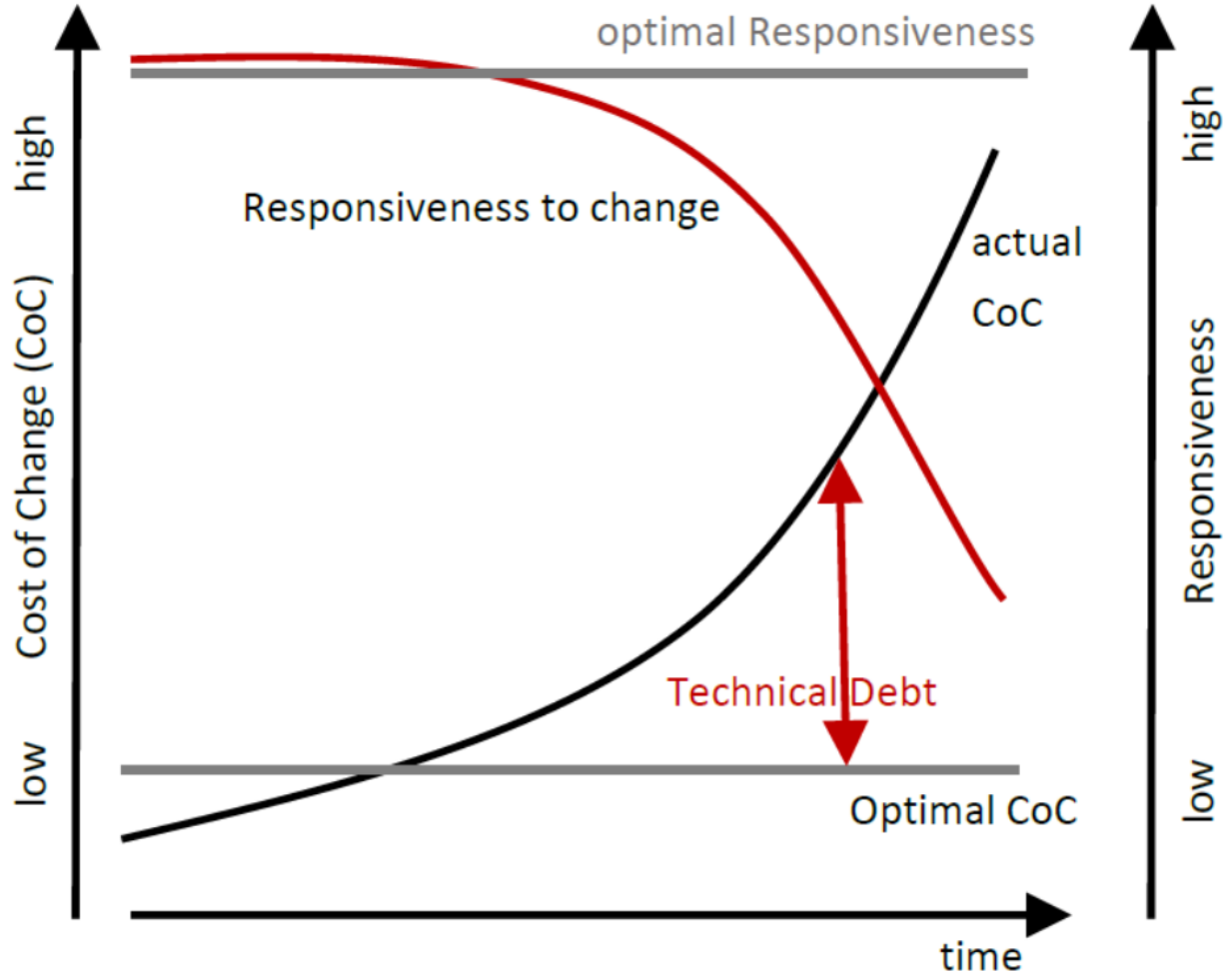
Clean Code란 ?

"모든 팀원이 **이해(understandability)하기 쉽도록** 작성된 코드"

- 의존성이 적어 **단순하고**, 테스트코드로 **검증된** 코드
- 코드를 해석하는 시간과 수정하는 시간의 비율은 10(read) : 1 (write)
- 대부분의 결함은 기존 코드 수정 시에 발생되므로 이해하기 쉬운 코드는 오류의 위험을 최소화함

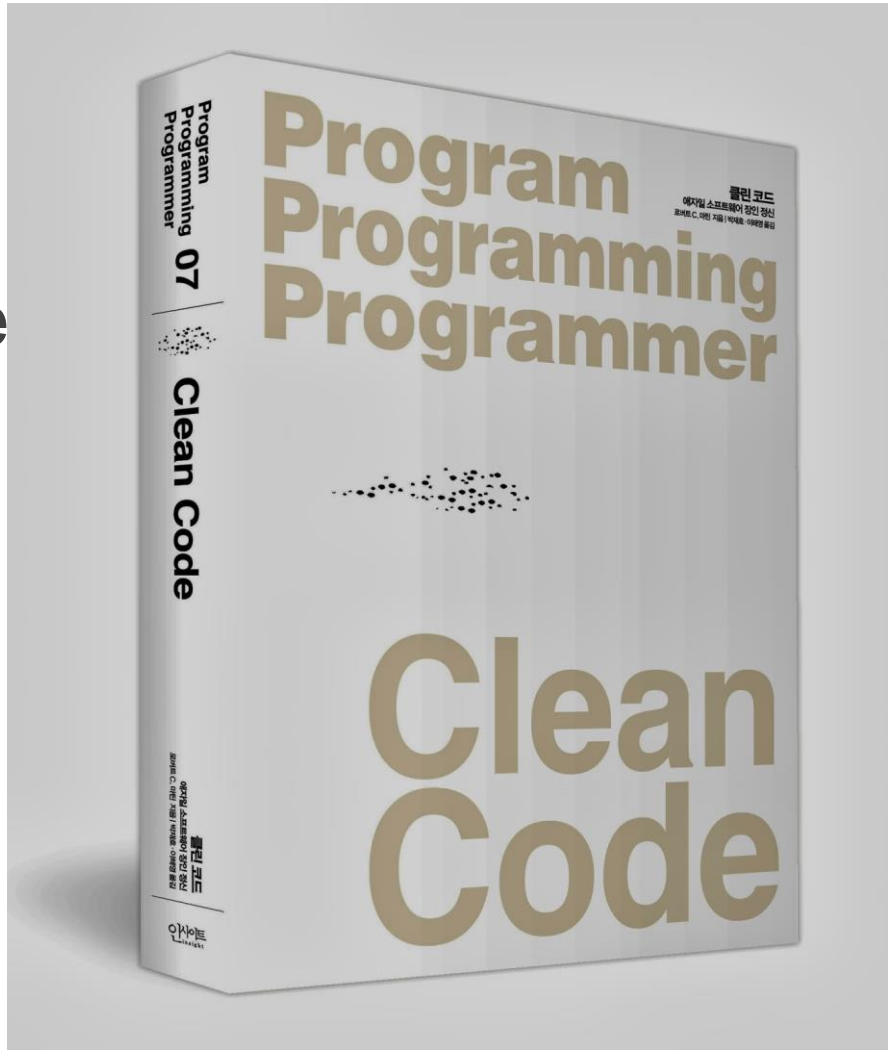
Readability, Changeability, Extensibility, Maintainability…

Clean Code를 작성해야 하는 이유



Clean Code의 주요 영역

Naming
Comments
Data Structure
Method
Style
Generics
Concurrency
Class
Error Handling
Annotation
Performance
Unit Test
Architecture

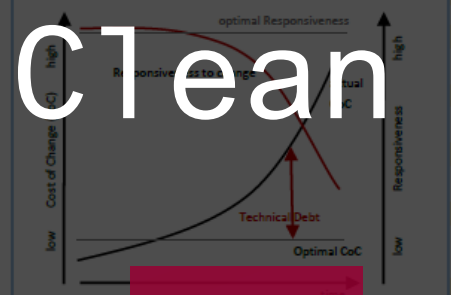


Clean Code의 주요원칙 (General)

Follow Standard Conventions	Coding 표준, 아키텍처 표준 및 설계 가이드를 준수하라.
Keep It Simple, Stupid (KISS)	단순한 것이 효율적이며, 복잡함을 최소화하라.
Boy Scout Rule	캠핑장을 떠나기 전에 원래보다 깨끗하게 해야 한다. (참조되거나 수정되는 코드는 원래보다 clean하게 해야 함)
Root Cause Analysis	항상 근본적인 원인을 찾아라. 그렇지 않으면 반복될 것이다
Do Not Multiple Languages in One Source File	하나의 언어는 하나의 파일로 작성(Java, JavaScript, Html…)

출처 : Clean Code Cheat Sheet https://www.planetgeek.ch/wp-content/uploads/2014/11/Clean-Code-V2.4.pdf

Clean Code의 주요원칙 (Class Design)



S Simple Responsibility Principle (SRP)

하나의 클래스는 하나의 책임만 가져야 한다.

O Open/Closed Principle (OCP)

클래스는 확장에 대하여 열려 있어야 하고, 변경에 대해서는 닫혀 있어야 한다.

L Liskov Substitution Principle (LSP)

파생 클래스의 메소드는 기반 클래스의 메서드를 대체하여 사용될 수 있어야 한다.

I Interface Segregation Principle (ISP)

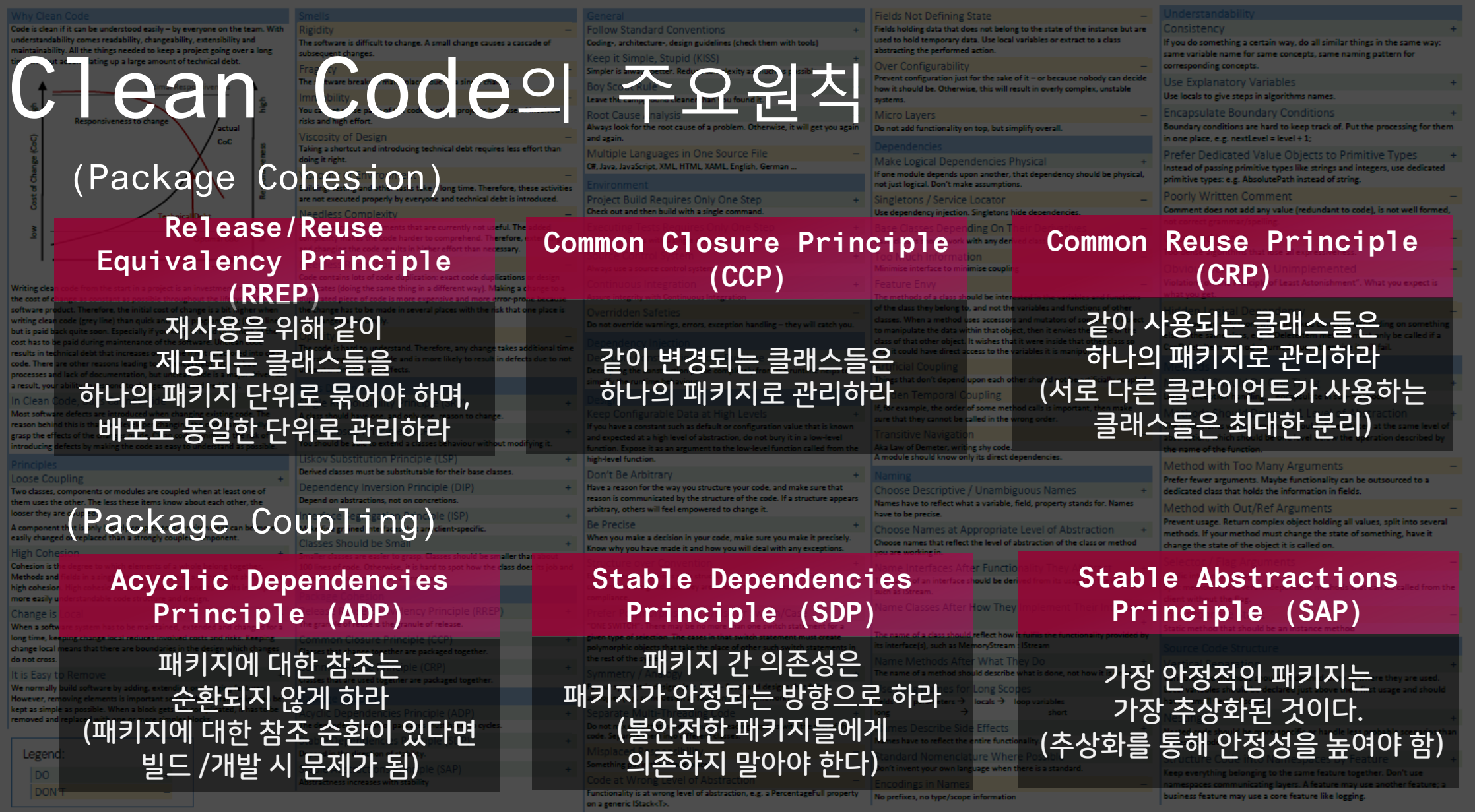
클라이언트가 사용하지 않는 메소드에 의존하지 않아야 한다.

D Dependency Inversion Principle (DIP)

추상화된 것은 구체적인 것에 의존하면 안된다. (자주 변경되는 구체적인 것에 의존하지 말고 추상화된 것을 참조)

출처 : Clean Code Cheat Sheet https://www.planetgeek.ch/wp-content/uploads/2014/11/Clean-Code-V2.4.pdf

Clean Code의 주요원칙



Release/Reuse Equivalency Principle (RREP)

재사용을 위해 같이
제공되는 클래스들을
하나의 패키지 단위로 묶어야 하며,
배포도 동일한 단위로 관리하라

Common Closure Principle (CCP)

같이 변경되는 클래스들은
하나의 패키지로 관리하라

Common Reuse Principle (CRP)

같이 사용되는 클래스들은
하나의 패키지로 관리하라
(서로 다른 클라이언트가 사용하는
클래스들은 최대한 분리)

(Package Cohesion)

(Package Coupling)

Acyclic Dependencies Principle (ADP)

패키지에 대한 참조는
순환되지 않게 하라
(패키지에 대한 참조 순환이 있다면
빌드 /개발 시 문제가 됨)

Stable Dependencies Principle (SDP)

패키지 간 의존성은
패키지가 안정되는 방향으로 하라
(불안전한 패키지들에게
의존하지 말아야 한다)

Stable Abstractions Principle (SAP)

가장 안정적인 패키지는
가장 추상화된 것이다.
(추상화를 통해 안정성을 높여야 함)

출처 : Clean Code Cheat Sheet https://www.planetgeek.ch/wp-content/uploads/2014/11/Clean-Code-V2.4.pdf

쉬어가는 퀴즈 타임!

Q. java.util.Arrays.binarySearch() 메소드입니다. 이 코드는 언제나 잘 동작할까요?

```java
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1;
        else if (midVal > key)
            high = mid - 1;
        else
            return mid; // key found
    }
    return -(low + 1); // key not found.
}
```

쉬어가는 퀴즈 타임!

Q. java.util.Arrays.binarySearch() 메소드입니다. 이 코드는 언제나 잘 동작할까요?

```java
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1;
        else if (midVal > key)
            high = mid - 1;
        else
            return mid; // key found
    }
    return -(low + 1); // key not found.
}
```

```java
int mid = low + ((high - low) / 2);
//또는
int mid = (low + high) >>> 1;
```

Binary Search 알고리즘은 1946년 만들어졌는데,
해당 버그는 20년이 지난 1962년에 발견됨.

2의 30승 이상의 원소를 갖는 경우 버그가 발생.

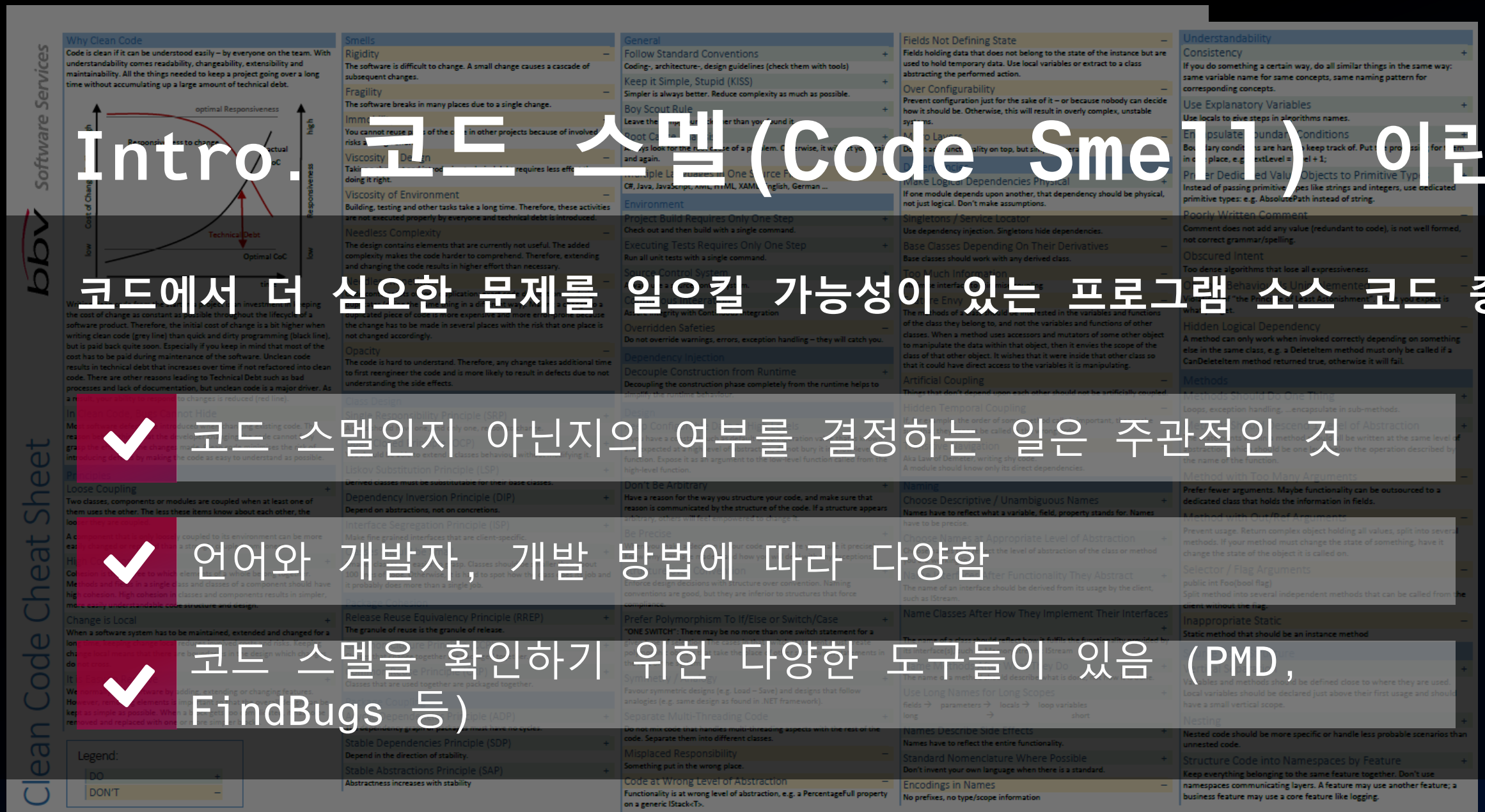
심지어, JDK bug 수정도 2006년에서야 적용되었고,
아직도 많은 언어들이 해당 버그를 갖고 있음

Clean Code Refactoring

Intro. 코드 스멜(Code Smell)이란?

코드에서 더 심오한 문제를 일으킬 가능성이 있는 프로그램 소스 코드 증상

- ✔ 코드 스멜인지 아닌지의 여부를 결정하는 일은 주관적인 것
- ✔ 언어와 개발자, 개발 방법에 따라 다양함
- ✔ 코드 스멜을 확인하기 위한 다양한 도구들이 있음 (PMD, FindBugs 등)



1. Naming

의미 있는 이름을 사용하자.

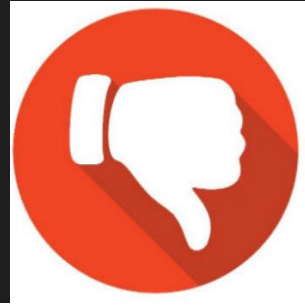
```
1   var id;     //사용자계정
2   var pwd;    //비밀번호
3   var sq;     //학번
4   var nm;     //이름
5   var em;     //이메일
6
7   for (var i = 0 ; i < users.length ; i++){
8       if(users[i].id == id && users[i].pwd == pwd){
9           alert(nm + "님. 환영합니다.")
10      }
11  }
```

```
1   var studentId;
2   var studentPassword;
3   var studentSequenceNumber;
4   var studentName;
5   var studentEmailAddress;
6
7   for (var i = 0 ; i < users.length ; i++){
8       if(users[i].studentId == studentId
9           && users[i].studentPassword === studentPassword){
10              alert(studentName + "님. 환영합니다.")
11          }
12  }
```

2. Style

적절한 행 길이와 들여 쓰기를 유지하고, 빈행으로 분리하라.

```
1  package fitnesse.wikitext.widgets;
2  import java.util.regex.*;
3  public class BoldWidget extends ParentWidget {
4  public static final String REGEXP = "'''.+?'''";
5  private static final Pattern pattern =
6  Pattern.compile("'''(.+?)'''", Pattern.MULTILINE + Pattern.DOTALL);
7  public BoldWidget(ParentWidget parent, String text) throws Exception {
8  super(parent);
9  Matcher match = pattern.matcher(text); match.find();
10 addChildWidgets(match.group(1));}
11 public String render() throws Exception {
12 StringBuffer html = new StringBuffer("<b>");
13 html.append(childHtml()).append("</b>");
14 return html.toString();
15 }
16 }
```



```
1  package fitnesse.wikitext.widgets;
2
3  import java.util.regex.*;
4
5  public class BoldWidget extends ParentWidget {
6      public static final String REGEXP = "'''.+?'''";
7      private static final Pattern pattern =
8          Pattern.compile("'''(.+?)'''", Pattern.MULTILINE + Pattern.DOTALL
9      );
10
11     public BoldWidget(ParentWidget parent, String text) throws Exception {
12         super(parent);
13         Matcher match = pattern.matcher(text); match.find();
14         addChildWsssidgets(match.group(1));
15     }
16
17     public String render() throws Exception {
18         StringBuffer html = new StringBuffer("<b>");
19         html.append(childHtml()).append("</b>");
20         return html.toString();
21     }
22 }
```

변수는 사용하는 위치에 최대한 가까이 선언하고,
종속관계의 함수는 세로로 가까이 배치하라.

3. 주석

주석으로 나쁜 코드를 보완하지 말고, 코드로 의도를 표현하라.

```
1  //신입생이 등록금과 학생회비를 납부하고 수강신청을 완료하였는지 확인한다.
2  if(student.completeTutionPayment &&
3     student.completeStudentDues &&
4     student.applyEnrolment){
5
6  }
```

```
1  if(student.completeRegisterEntrance){
2
3  }
```

나쁜 주석과 좋은 주석을 구분 하기

4. Dead Code

사용하지 않는 코드

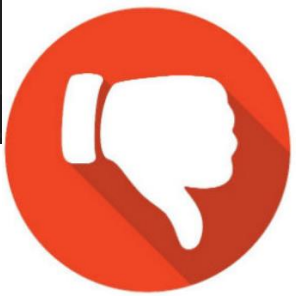
```java
public void howToDoInJava_method()
{
    System.out.println("how to do");

    if (true)
    {
        return;
    }
    else
    {
        return;
    }

    System.out.println("in java");
}
```

```java
// this is the unit which can produce the currentItem
Unit producer = getProducer(currentItem.metaType, seedPosition, currentItem.producerID);

// BasicBot 1.1 Patch End ////////////////////////////////////////////////

/*
    * if (currentItem.metaType.isUnit() &&
    * currentItem.metaType.getUnitType().isBuilding()) { if (producer
    * != null) { System.out.println("Build " +
    * currentItem.metaType.getName() + " producer : " +
    * producer.getType() + " ID : " + producer.getID()); } else {
    * System.out.println("Build " + currentItem.metaType.getName() +
    * " producer null"); } }
    */

Unit secondProducer = null;
boolean canMake = false;
```

5. Method

함수에서 인수의 개수는 적을수록 좋다

```
1  function createStudent(studentId, studentPassword, studentSequenceNumber, studentName, studentEmailAddress){
2      //신입생 정보 등록
3  }
4
5  function updateStudent(studentId, studentPassword, studentSequenceNumber, studentName, studentEmailAddress){
6      //신입생 정보 수정
7  }
8
9  function transStudent(studentId, studentPassword, studentSequenceNumber, studentName, studentEmailAddress){
10     //신입생 정보 교환
11 }
12
13 createStudent('gildong', 'password', '19052001', '홍길동', 'gildong@gmail.com');
```



```
1  Var student = {
2      studentId : 'gildong'
3      studentPassword : 'password'
4      studentSequenceNumber :'19052001'
5      studentName : '홍길동'
6      studentEmailAddress : 'gildong@gmail.com'
7  }
8
9  createStudent(student);
```

5. Method

최대한 단순하게 처리하자

```
function isAvaiableGiveSchoolDrivingLicense(student){
    if (student.age < 18) {
        return false;
    } else {
        return true;
    }
}
```

```
function isAvaiableGiveSchoolDrivingLicense(student){
    return student.age >= 18;
}
```

5. Method

단일 역할을 수행하도록 한다. 의도하지 않은 결과를 초래하지 않도록 하

```
1 function updateMobilePhoneNumbers(numberFirst, numberSecond){
2     smsService.sendSms("전화번호 변경이 완료되었습니다.");
3     return "010" + numberFirst + numberSecond;
4 }
```

```
1 function updateMobilePhoneNumbers(numberFirst, numberSecond){
2     return "010" + numberFirst + "-" + numberSecond;
3 }
```


THANK YOU ☺