Function Structure

Daum Corp.
백명석

목차

- Arguments
  - 개수 / Boolean
  - Innies not Outies / Null Defense
- Stepdown Rule
- Switches and Cases
- Temporal Coupling
- CQS
- Tell, Don't Ask
- Law of Demeter

Arguments

Arguments

- 함수의 인자는 복잡도를 증가한다

Arguments

- 함수의 인자는 복잡도를 증가한다
- Double Takes

Arguments - Three Arguments Max

Arguments - Three Arguments Max

- 2개가 거의 최대이다. 드물게 3개를 사용.

Arguments - Three Arguments Max

- 2개가 거의 최대이다. 드물게 3개를 사용.
- 인자가 3개 이상이면

Arguments - Three Arguments Max

- 2개가 거의 최대이다. 드물게 3개를 사용.
- 인자가 3개 이상이면
  - 순서를 외우기 어렵다.

Arguments - Three Arguments Max

- 2개가 거의 최대이다. 드물게 3개를 사용.
- 인자가 3개 이상이면
  - 순서를 외우기 어렵다.
- 3개가 넘는 인자가 함수에 전달되어야 한다면

Arguments - Three Arguments Max

- 2개가 거의 최대이다. 드물게 3개를 사용.
- 인자가 3개 이상이면
  - 순서를 외우기 어렵다.
- 3개가 넘는 인자가 함수에 전달되어야 한다면
  - 그 인자들은 객체여야 함.

Arguments - Three Arguments Max

- 2개가 거의 최대이다. 드물게 3개를 사용.
- 인자가 3개 이상이면
  - 순서를 외우기 어렵다.
- 3개가 넘는 인자가 함수에 전달되어야 한다면
  - 그 인자들은 객체여야 함.
- 생성자의 많은 수의 인자를 넘겨야 한다면

Arguments - Three Arguments Max

- 2개가 거의 최대이다. 드물게 3개를 사용.
- 인자가 3개 이상이면
  - 순서를 외우기 어렵다.
- 3개가 넘는 인자가 함수에 전달되어야 한다면
  - 그 인자들은 객체여야 함.
- 생성자의 많은 수의 인자를 넘겨야 한다면
  - 잘 지어진 이름을 갖는 setter가 낫다.

Arguments - Three Arguments Max

- 2개가 거의 최대이다. 드물게 3개를 사용.
- 인자가 3개 이상이면
  - 순서를 외우기 어렵다.
- 3개가 넘는 인자가 함수에 전달되어야 한다면
  - 그 인자들은 객체여야 함.
- 생성자의 많은 수의 인자를 넘겨야 한다면
  - 잘 지어진 이름을 갖는 setter가 낫다.
  - 아주 짧은 시간 동안은 객체가 완전하지 않을 수 있다.

Arguments - Three Arguments Max

- 2개가 거의 최대이다. 드물게 3개를 사용.
- 인자가 3개 이상이면
  - 순서를 외우기 어렵다.
- 3개가 넘는 인자가 함수에 전달되어야 한다면
  - 그 인자들은 객체여야 함.
- 생성자의 많은 수의 인자를 넘겨야 한다면
  - 잘 지어진 이름을 갖는 setter가 낫다.
  - 아주 짧은 시간 동안은 객체가 완전하지 않을 수 있다.
  - 단위 테스트로 이런 것이 문제가 되지 않도록

Arguments - No Boolean Arguments

Arguments - No Boolean Arguments

- Boolean을 인자로 갖는 함수는

Arguments - No Boolean Arguments

- Boolean을 인자로 갖는 함수는
  - 명확히 2가지 이상이 일을 하는 것

Arguments - No Boolean Arguments

- Boolean을 인자로 갖는 함수는
  - 명확히 2가지 이상이 일을 하는 것
  - true 경우를 위한 일 + false 경우를 위한 일

Arguments - No Boolean Arguments

- Boolean을 인자로 갖는 함수는
  - 명확히 2가지 이상이 일을 하는 것
  - true 경우를 위한 일 + false 경우를 위한 일
  - 2개의 함수로 분리

Arguments - No Boolean Arguments

- Boolean을 인자로 갖는 함수는
  - 명확히 2가지 이상이 일을 하는 것
  - true 경우를 위한 일 + false 경우를 위한 일
  - 2개의 함수로 분리
- Boolean 인자는 오류/오해를 유발

Arguments - No Boolean Arguments

- Boolean을 인자로 갖는 함수는
  - 명확히 2가지 이상이 일을 하는 것
  - true 경우를 위한 일 + false 경우를 위한 일
  - 2개의 함수로 분리
- Boolean 인자는 오류/오해를 유발
- 2개의 Boolean 인자를 받는 함수

Arguments - No Boolean Arguments

- Boolean을 인자로 갖는 함수는
  - 명확히 2가지 이상이 일을 하는 것
  - true 경우를 위한 일 + false 경우를 위한 일
  - 2개의 함수로 분리
- Boolean 인자는 오류/오해를 유발
- 2개의 Boolean 인자를 받는 함수
  - 4가지 일을 하는 것

Arguments - No Boolean Arguments

- Boolean을 인자로 갖는 함수는
  - 명확히 2가지 이상이 일을 하는 것
  - true 경우를 위한 일 + false 경우를 위한 일
  - 2개의 함수로 분리
- Boolean 인자는 오류/오해를 유발
- 2개의 Boolean 인자를 받는 함수
  - 4가지 일을 하는 것
- 순서는 더 복잡

Arguments - No Boolean Arguments

- Boolean을 인자로 갖는 함수는
  - 명확히 2가지 이상이 일을 하는 것
  - true 경우를 위한 일 + false 경우를 위한 일
  - 2개의 함수로 분리
- Boolean 인자는 오류/오해를 유발
- 2개의 Boolean 인자를 받는 함수
  - 4가지 일을 하는 것
- 순서는 더 복잡
  - true, false가 전달되면 무슨 의미인지 알기 어렵고,

Arguments - No Boolean Arguments

- Boolean을 인자로 갖는 함수는
  - 명확히 2가지 이상이 일을 하는 것
  - true 경우를 위한 일 + false 경우를 위한 일
  - 2개의 함수로 분리
- Boolean 인자는 오류/오해를 유발
- 2개의 Boolean 인자를 받는 함수
  - 4가지 일을 하는 것
- 순서는 더 복잡
  - true, false가 전달되면 무슨 의미인지 알기 어렵고,
  - 순서가 무슨 의미를 갖는지 알기 어렵다.

Arguments - Innies not Outies

Arguments - Innies not Outies

- No output arguments please

Arguments - Innies not Outies

- No output arguments please
  - Argument는 함수로 전달되는 것이지,

Arguments - Innies not Outies

- No output arguments please
  - Argument는 함수로 전달되는 것이지,
  - 함수로부터 변경되어 나오는 것이라고는 생각하지 않음

Arguments - Innies not Outies

- No output arguments please
  - Argument는 함수로 전달되는 것이지,
  - 함수로부터 변경되어 나오는 것이라고는 생각하지 않음
  - output argument 대신 return value로 처리

Arguments - Innies not Outies

- No output arguments please
  - Argument는 함수로 전달되는 것이지,
  - 함수로부터 변경되어 나오는 것이라고는 생각하지 않음
  - output argument 대신 return value로 처리

```
1  // does 'returnText' go in ? Go out ? Both ?
2  private String toSimpleText(Parse table, StringBuffer returnText) {
3      if(table.parts == null) {
4          simpleTextOfLeave(table, returnText);
5          simpleTextOfMore(table, returnText);
6          return returnText.toString();
7      }
8      simpleTextOfParts(table, returnText);
9      simpleTextOfMore(table, returntext);
10     return returntext.toString();
11 }
```

  - output arguments cause double takes

Arguments - The null defense

Arguments - The null defense

- null을 전달/기대하는 함수는

Arguments - The null defense

- null을 전달/기대하는 함수는
  - boolean을 전달하는 만큼 잘못된 것

Arguments - The null defense

- null을 전달/기대하는 함수는
  - boolean을 전달하는 만큼 잘못된 것
  - null인 경우의 행위 + null이 아닌 경우의 행위

Arguments - The null defense

- null을 전달/기대하는 함수는
  - boolean을 전달하는 만큼 잘못된 것
  - null인 경우의 행위 + null이 아닌 경우의 행위
  - 2개의 함수를 만드는 것이 맞다.

Arguments - The null defense

- null을 전달/기대하는 함수는
  - boolean을 전달하는 만큼 잘못된 것
  - null인 경우의 행위 + null이 아닌 경우의 행위
  - 2개의 함수를 만드는 것이 맞다.
- null을 pseudo boolean로 쓰지 마라.

Arguments - The null defense

- null을 전달/기대하는 함수는
  - boolean을 전달하는 만큼 잘못된 것
  - null인 경우의 행위 + null이 아닌 경우의 행위
  - 2개의 함수를 만드는 것이 맞다.
- null을 pseudo boolean로 쓰지 마라.
- defensive programming을 지양

Arguments - The null defense

- null을 전달/기대하는 함수는
  - boolean을 전달하는 만큼 잘못된 것
  - null인 경우의 행위 + null이 아닌 경우의 행위
  - 2개의 함수를 만드는 것이 맞다.
- null을 pseudo boolean로 쓰지 마라.
- defensive programming을 지양
  - 코드를 null, 에러 체크로 더럽히지 말라.

Arguments - The null defense

- null을 전달/기대하는 함수는
  - boolean을 전달하는 만큼 잘못된 것
  - null인 경우의 행위 + null이 아닌 경우의 행위
  - 2개의 함수를 만드는 것이 맞다.
- null을 pseudo boolean로 쓰지 마라.
- defensive programming을 지양
  - 코드를 null, 에러 체크로 더럽히지 말라.
  - 잘못되었다는 단서(smell)

Arguments - The null defense

- null을 전달/기대하는 함수는
  - boolean을 전달하는 만큼 잘못된 것
  - null인 경우의 행위 + null이 아닌 경우의 행위
  - 2개의 함수를 만드는 것이 맞다.
- null을 pseudo boolean로 쓰지 마라.
- defensive programming을 지양
  - 코드를 null, 에러 체크로 더럽히지 말라.
  - 잘못되었다는 단서(smell)
  - 팀원이나 단위 테스트를 못 믿는다는 말

Arguments - The null defense

- null을 전달/기대하는 함수는
  - boolean을 전달하는 만큼 잘못된 것
  - null인 경우의 행위 + null이 아닌 경우의 행위
  - 2개의 함수를 만드는 것이 맞다.
- null을 pseudo boolean로 쓰지 마라.
- defensive programming을 지양
  - 코드를 null, 에러 체크로 더럽히지 말라.
  - 잘못되었다는 단서(smell)
  - 팀원이나 단위 테스트를 못 믿는다는 말
  - null 여부를 지속적으로 조사할 것이 아니라

Arguments - The null defense

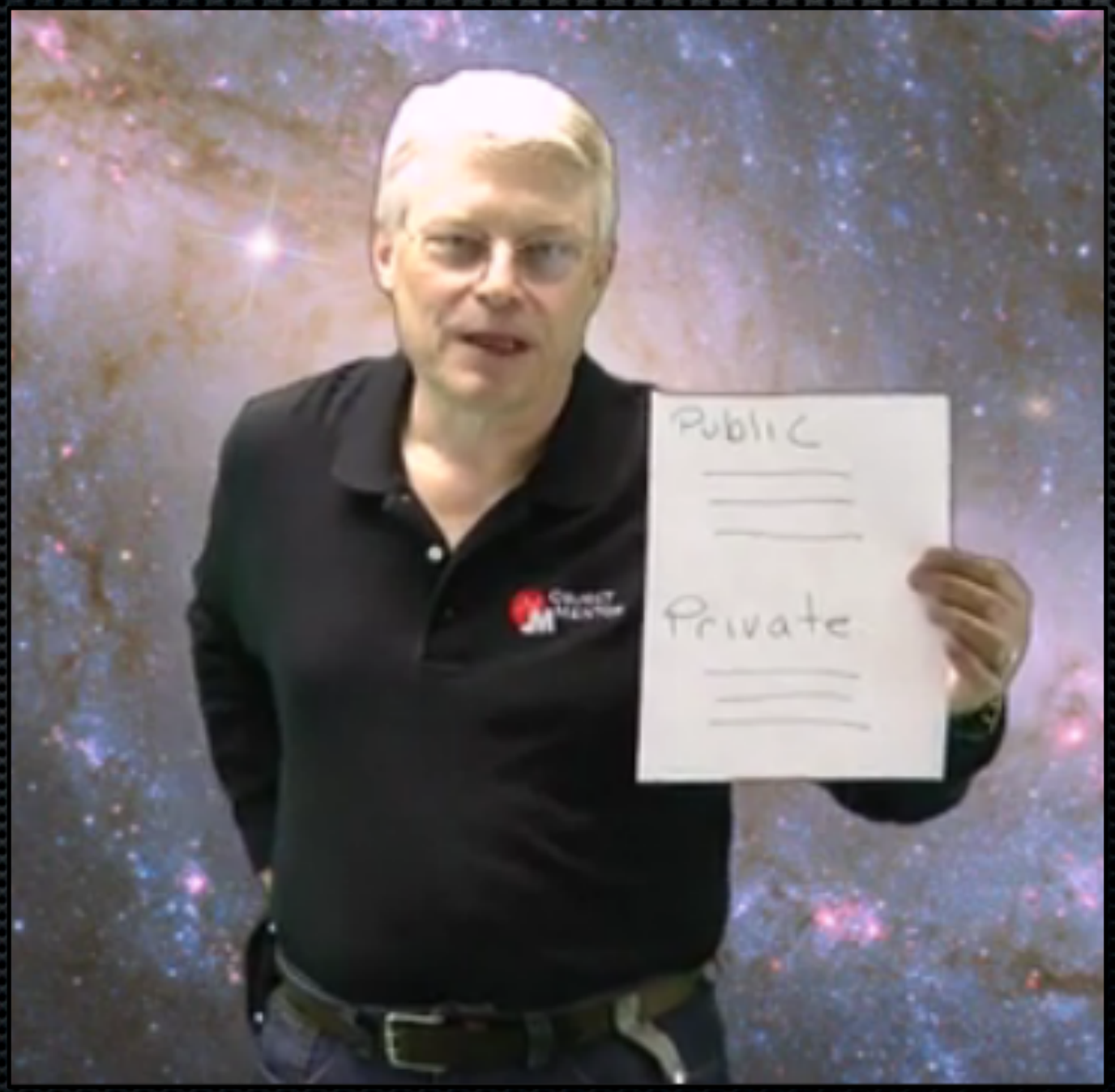
- null을 전달/기대하는 함수는
  - boolean을 전달하는 만큼 잘못된 것
  - null인 경우의 행위 + null이 아닌 경우의 행위
  - 2개의 함수를 만드는 것이 맞다.
- null을 pseudo boolean로 쓰지 마라.
- defensive programming을 지양
  - 코드를 null, 에러 체크로 더럽히지 말라.
  - 잘못되었다는 단서(smell)
  - 팀원이나 단위 테스트를 못 믿는다는 말
  - null 여부를 지속적으로 조사할 것이 아니라
  - 단위 테스트에서 검증해야

The Stepdown Rule

- 모든 public은 위에, 모든 private은 아래에

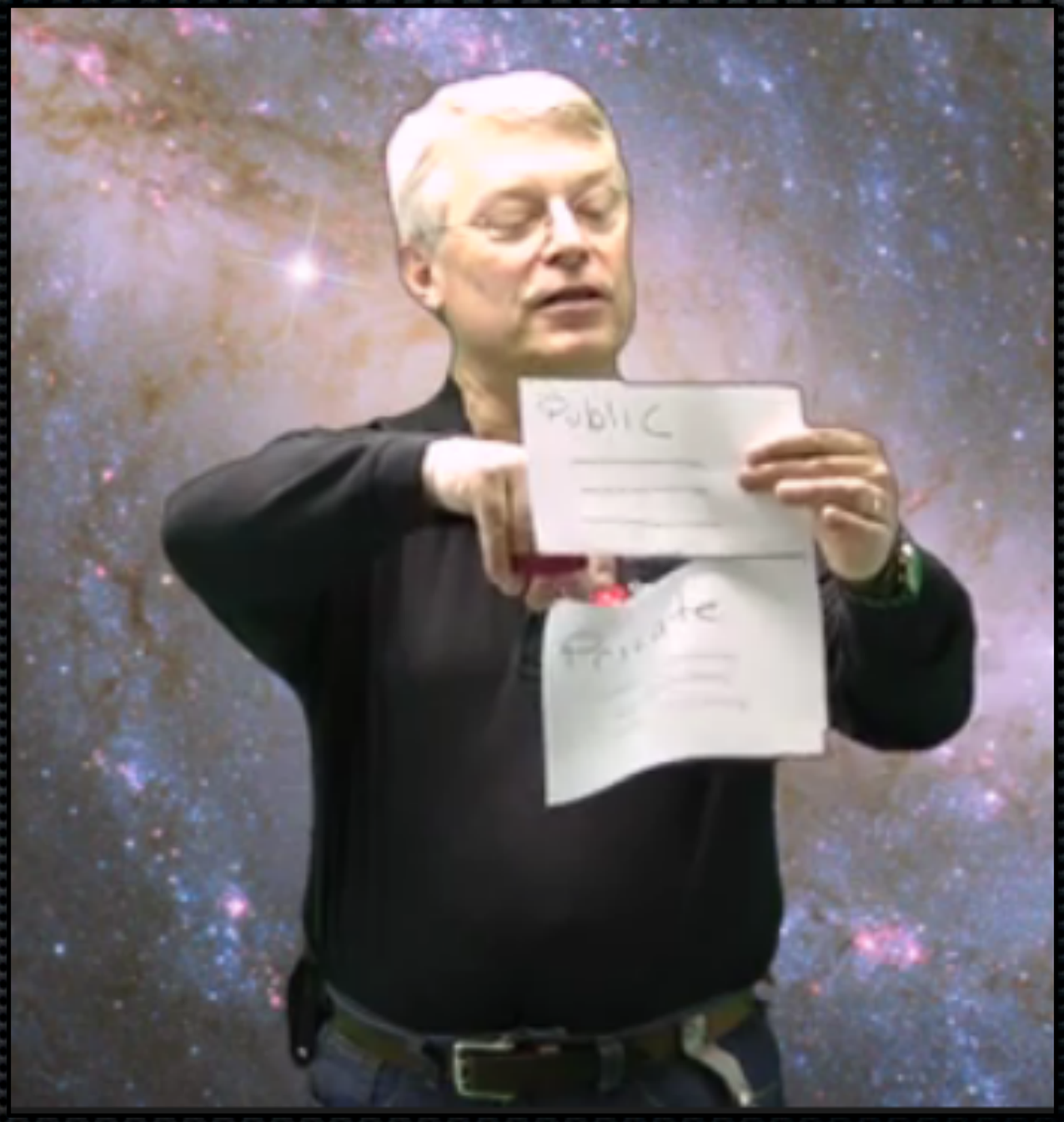
The Stepdown Rule

- 모든 public은 위에, 모든 private은 아래에



The Stepdown Rule

The Stepdown Rule



- public part만 사용자들에게 전달하면 됨.

The Stepdown Rule

The Stepdown Rule

- Important part goes to top,

The Stepdown Rule

- Important part goes to top,
- Details goes to bottom

The Stepdown Rule

- Important part goes to top,
- Details goes to bottom
  - 잡지의 예: 헤드라인, 시놉시스, 아웃라인, 상세 내역

The Stepdown Rule

- Important part goes to top,
- Details goes to bottom
  - 잡지의 예: 헤드라인, 시놉시스, 아웃라인, 상세 내역
- 이 방식의 2가지 이점

The Stepdown Rule

- Important part goes to top,
- Details goes to bottom
  - 잡지의 예: 헤드라인, 시놉시스, 아웃라인, 상세 내역
- 이 방식의 2가지 이점
  - 편집자들은 마지막 부분은 필수적인 내용 전달 오류 없이 제거할 수 있다.

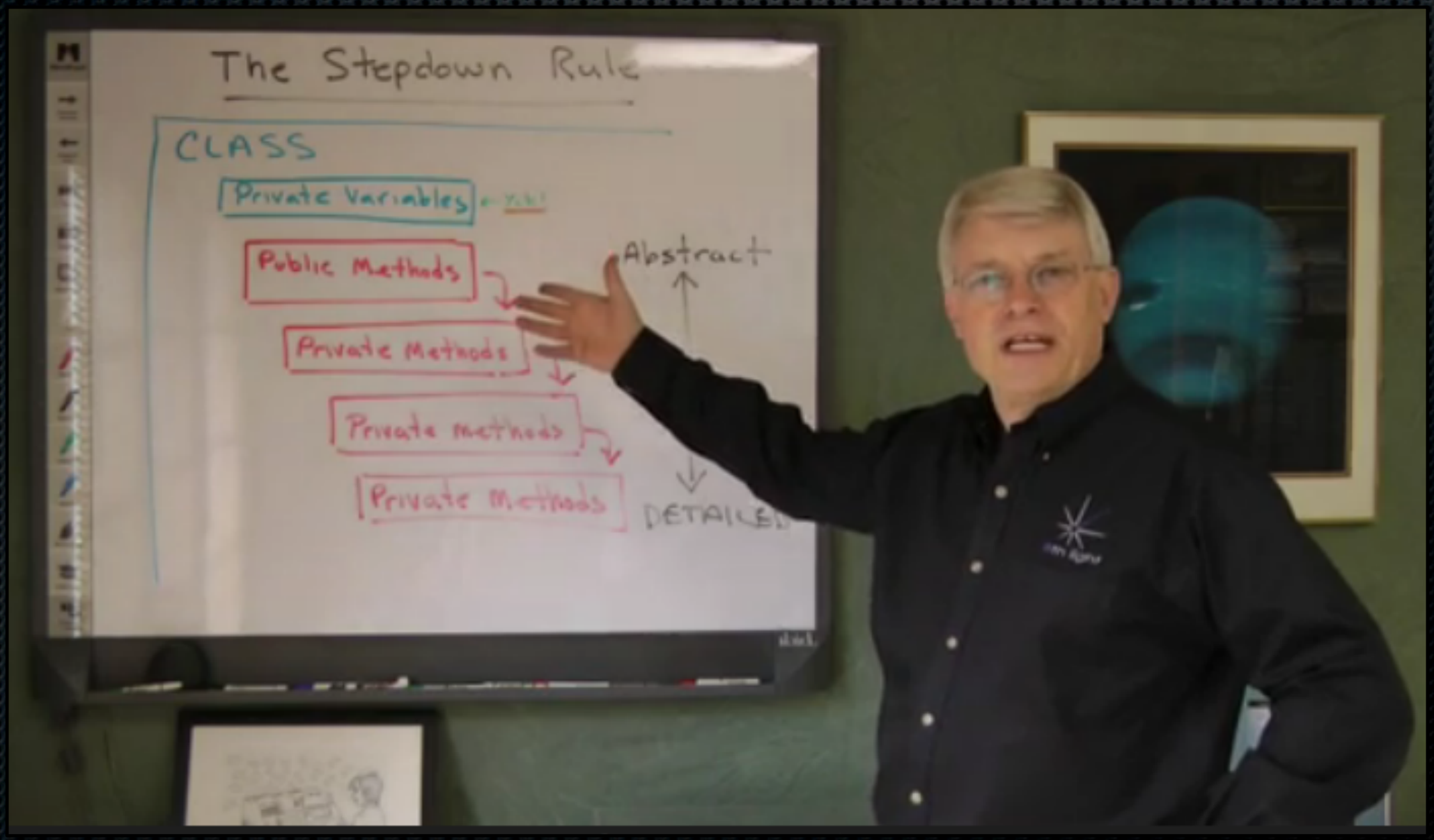
The Stepdown Rule

- Important part goes to top,
- Details goes to bottom
  - 잡지의 예: 헤드라인, 시놉시스, 아웃라인, 상세 내역
- 이 방식의 2가지 이점
  - 편집자들은 마지막 부분은 필수적인 내용 전달 오류 없이 제거할 수 있다.
  - 독자들은 제일 위에서부터 읽기 시작해서 지루해지면 그만 읽으면 된다.

The Stepdown Rule

The Stepdown Rule

- backward reference없이 top에서 bottom으로 읽을 수 있게



The Stepdown Rule

The Stepdown Rule

```
public void serve(Socket s) {
    try {
        tryProcessInstructions(s);
    } catch(Throwable e) {
    } finally {
        slimFactory.stop();
        close();
        closeEnclosingServiceInSeperateThread();
    }
}

    private void tryProcessInstructions(Socket s) throws Exception {
        initialize(s);
        ...
        more = processOneSetOfInstructions();
    }

        private initialize(Socket s) throws Exception {
            ...
        }

        private boolean processOneSetOfInstructions {
            String instructions = getInstructionFromClient();
            ...
            return processTheInstructions(instructions);
        }

            private String getInstructionFromClient() {
                ...
            }

            private boolean processTheInstructions(String instructions) {
                ...
                executeInstructions(instructions);
                ...
                sendResultsToClient(results);
            }

                private List<Object> executeInstructions(String instructions) {
                    ...
                }

                private void sendResultsToClient(List<Object> results) {
                    ...
                }

    private void close() {
        ...
    }

    private void closeEnclosingServiceInSeperateThread() {
        ...
    }
```

Switches and Cases

Switches and Cases

- switch 문장 사용을 왜 꺼리나 ?

Switches and Cases

- switch 문장 사용을 왜 꺼리나 ?
  - switch 문장은 OO 스럽지 않다 ?

Switches and Cases

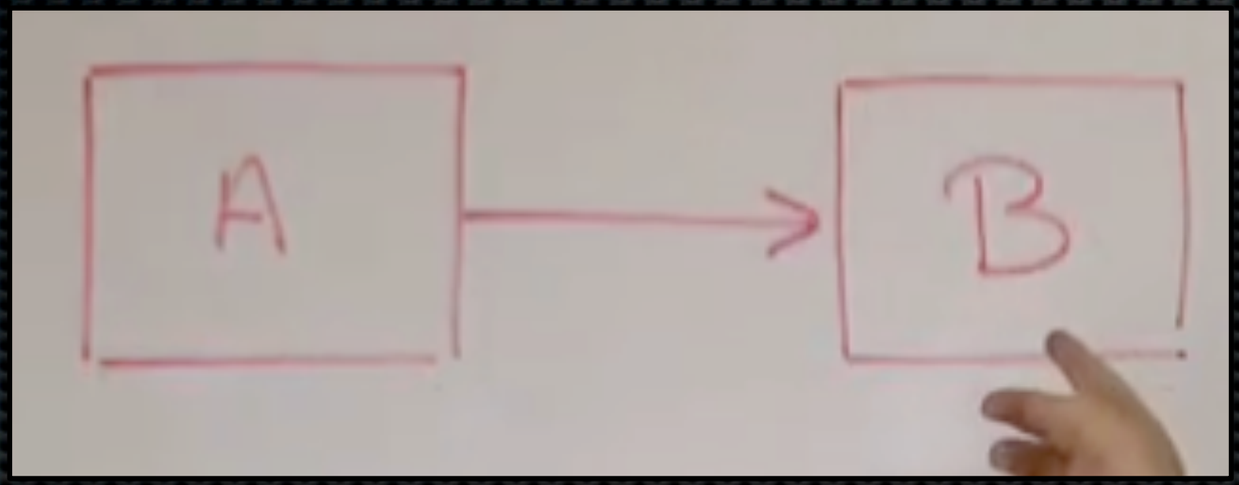
- switch 문장 사용을 왜 꺼리나 ?
  - switch 문장은 OO 스럽지 않다 ?
  - 만족할 만한 대답이 아니다.

Switches and Cases

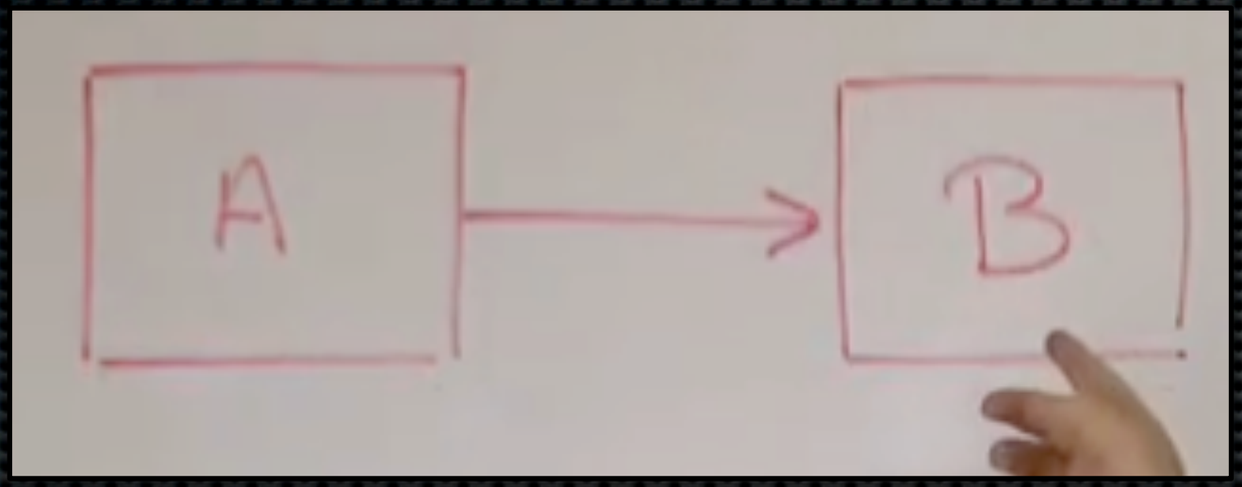
- switch 문장 사용을 왜 꺼리나 ?
  - switch 문장은 OO 스럽지 않다 ?
  - 만족할 만한 대답이 아니다.
  - OO의 가장 큰 이점 중 하나는 의존성 관리 능력이다.

Switches and Cases

Switches and Cases

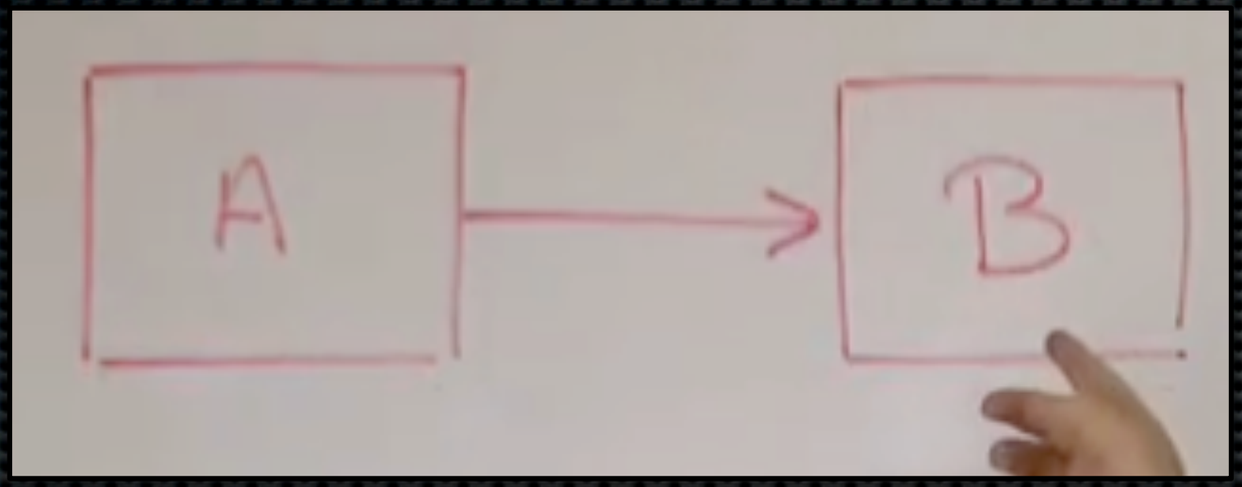


Switches and Cases



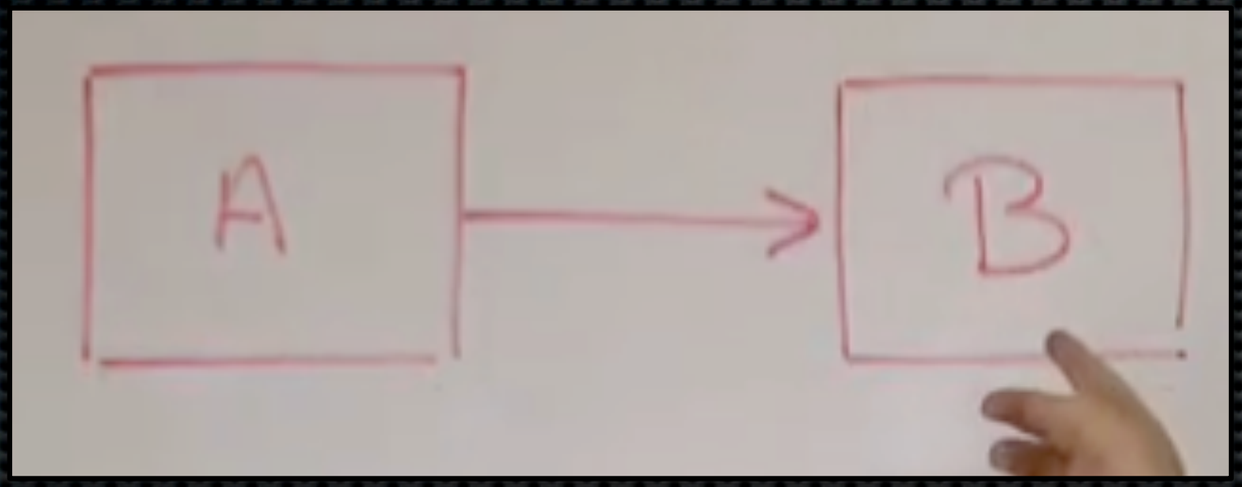
- 모듈 A와 모듈 B가 있다.

Switches and Cases



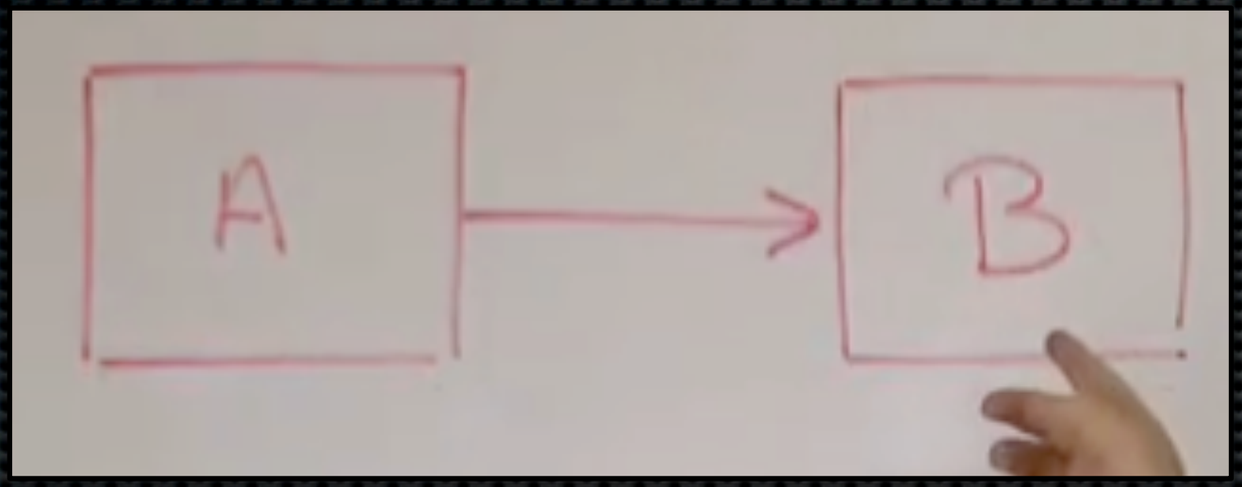
- 모듈 A와 모듈 B가 있다.
- 모듈 A의 함수가 모듈 B의 함수를 호출한다.

Switches and Cases



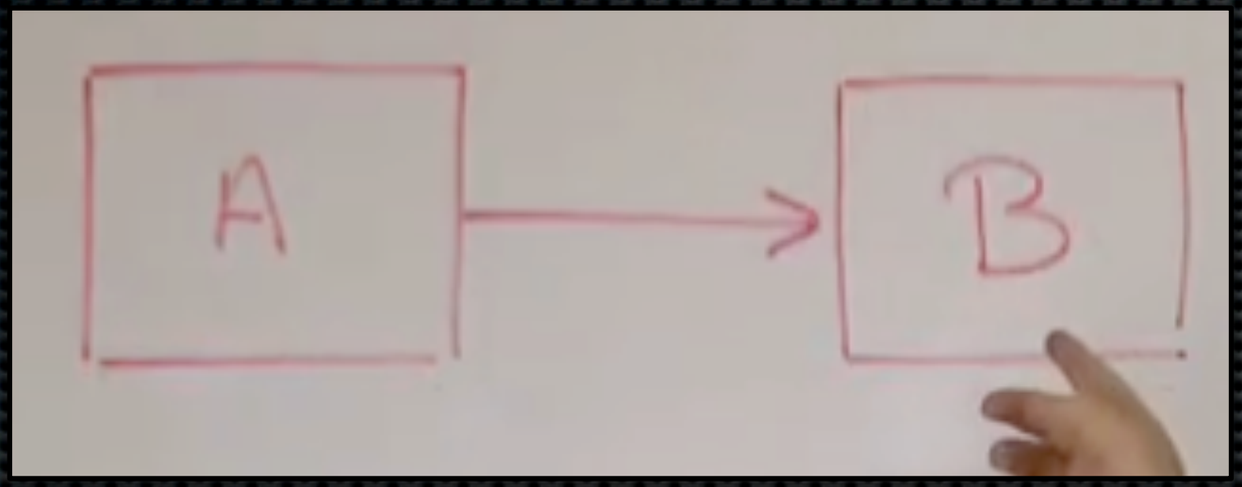
- 모듈 A와 모듈 B가 있다.
- 모듈 A의 함수가 모듈 B의 함수를 호출한다.
  - 모듈 A에서 모듈 B로의 의존성 관계가 있다.

Switches and Cases



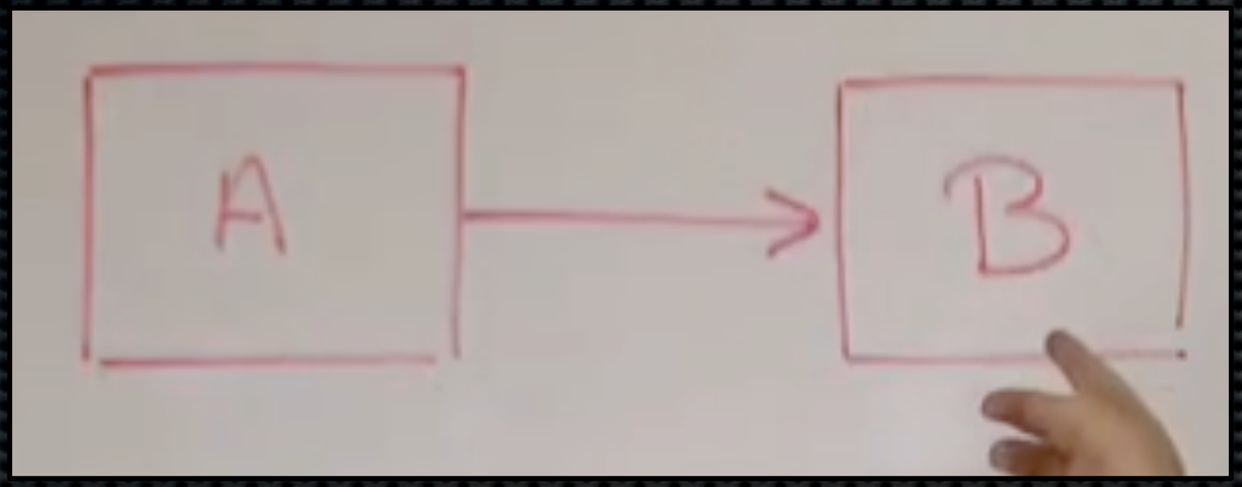
- 모듈 A와 모듈 B가 있다.
- 모듈 A의 함수가 모듈 B의 함수를 호출한다.
  - 모듈 A에서 모듈 B로의 의존성 관계가 있다.
- 의존성은 2개의 컴포넌트를 갖는다.

Switches and Cases



- 모듈 A와 모듈 B가 있다.
- 모듈 A의 함수가 모듈 B의 함수를 호출한다.
  - 모듈 A에서 모듈 B로의 의존성 관계가 있다.
- 의존성은 2개의 컴포넌트를 갖는다.
  - flow of control(RT 의존성): 모듈 A가 모듈 B를 호출

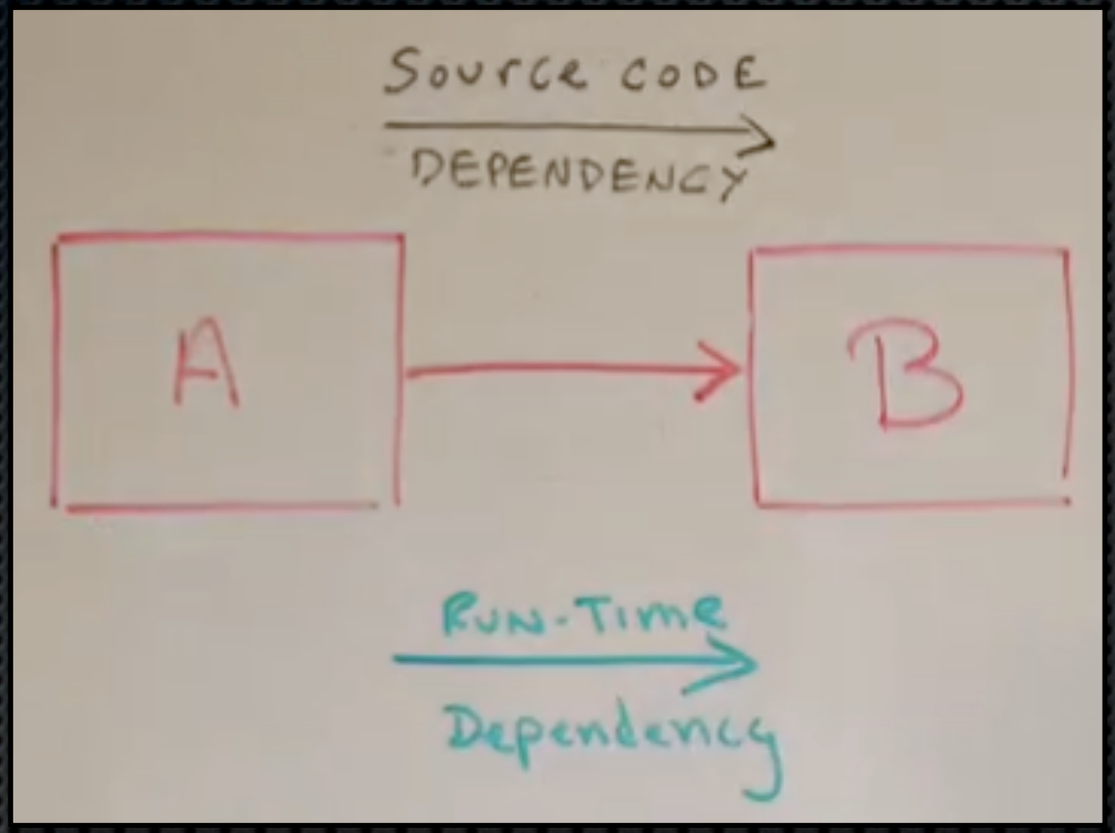
Switches and Cases



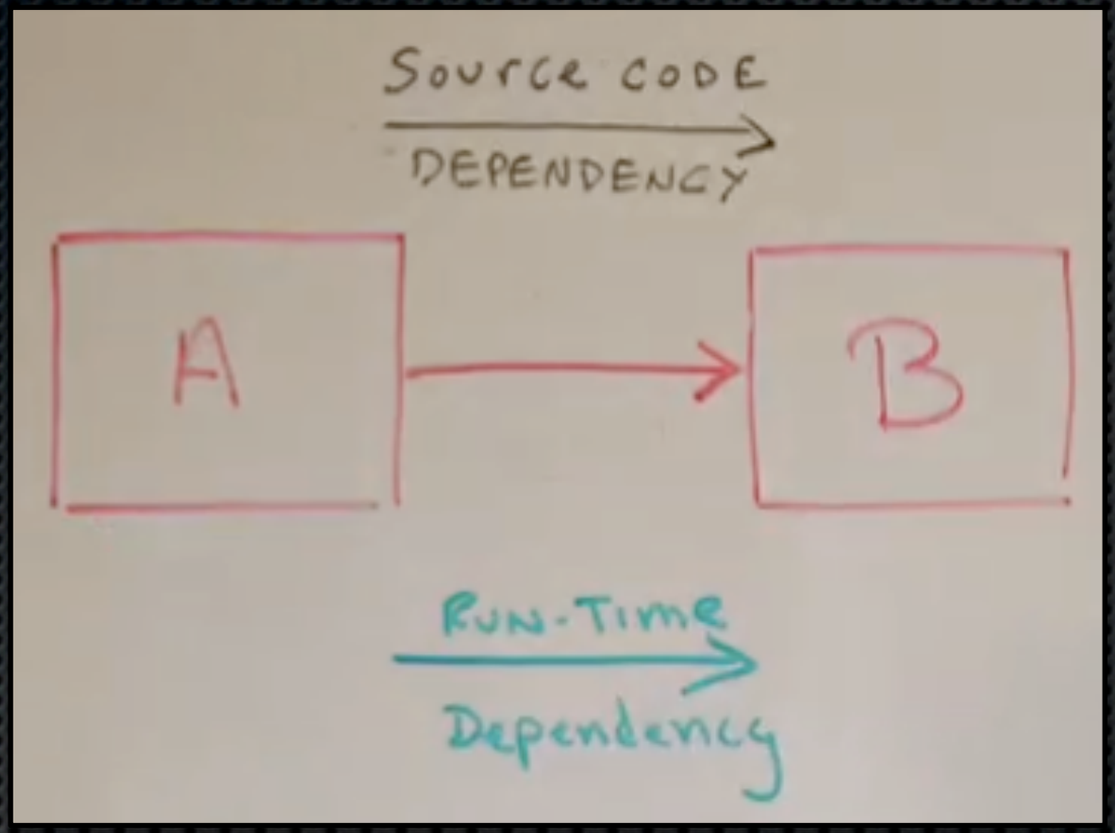
- 모듈 A와 모듈 B가 있다.
- 모듈 A의 함수가 모듈 B의 함수를 호출한다.
  - 모듈 A에서 모듈 B로의 의존성 관계가 있다.
- 의존성은 2개의 컴포넌트를 갖는다.
  - flow of control(RT 의존성): 모듈 A가 모듈 B를 호출
  - source code dependency(import 등)

Switches and Cases

Switches and Cases

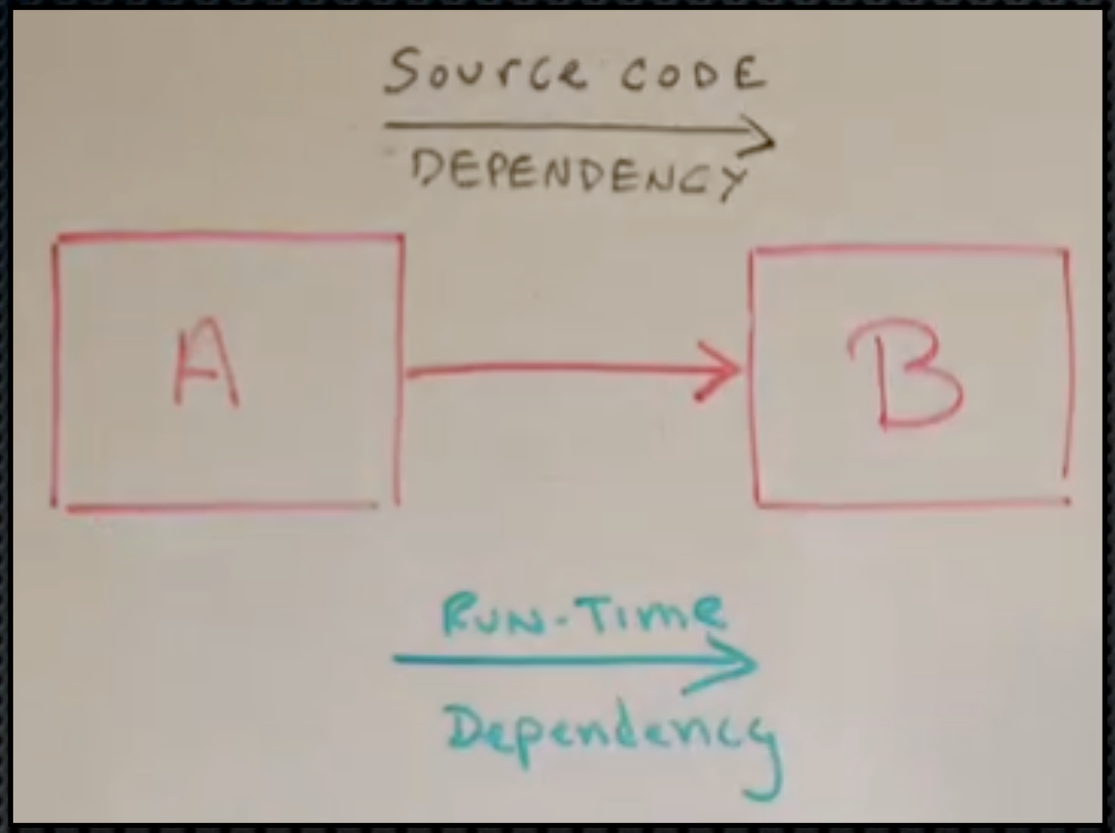


Switches and Cases



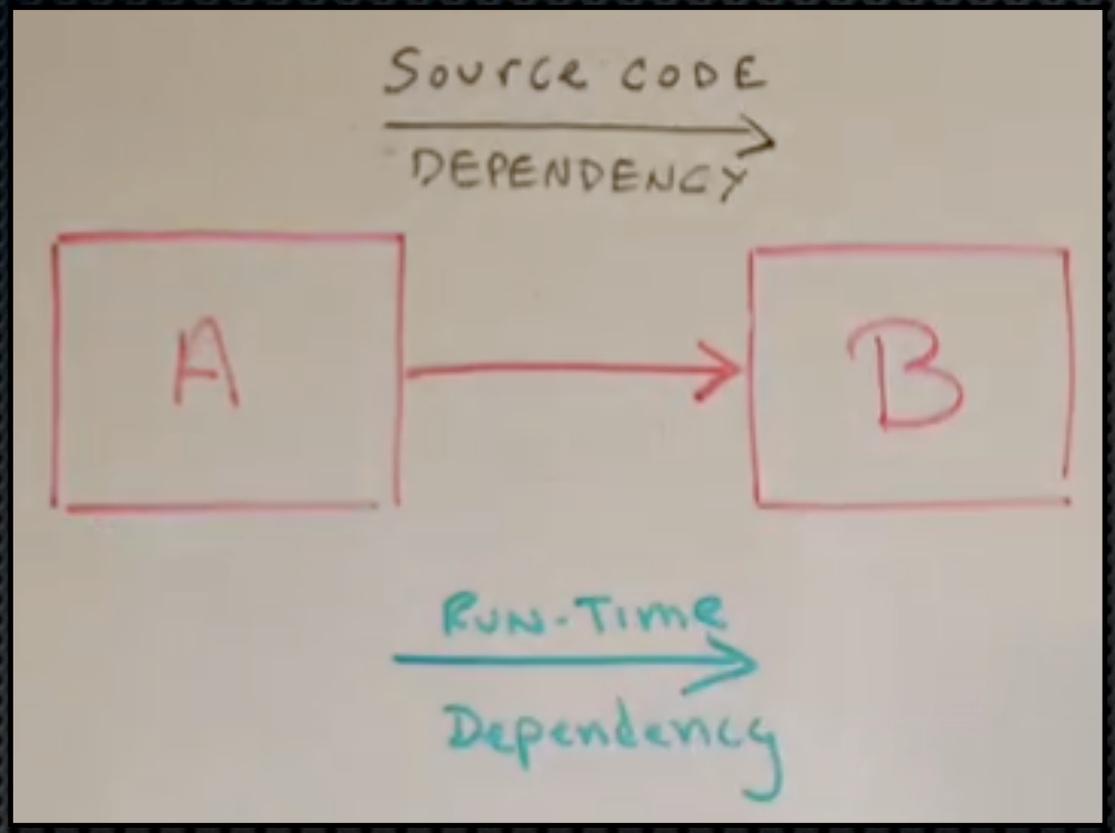
- 모듈 A와 B를 독립적으로 배포하고자 한다면 ?

Switches and Cases



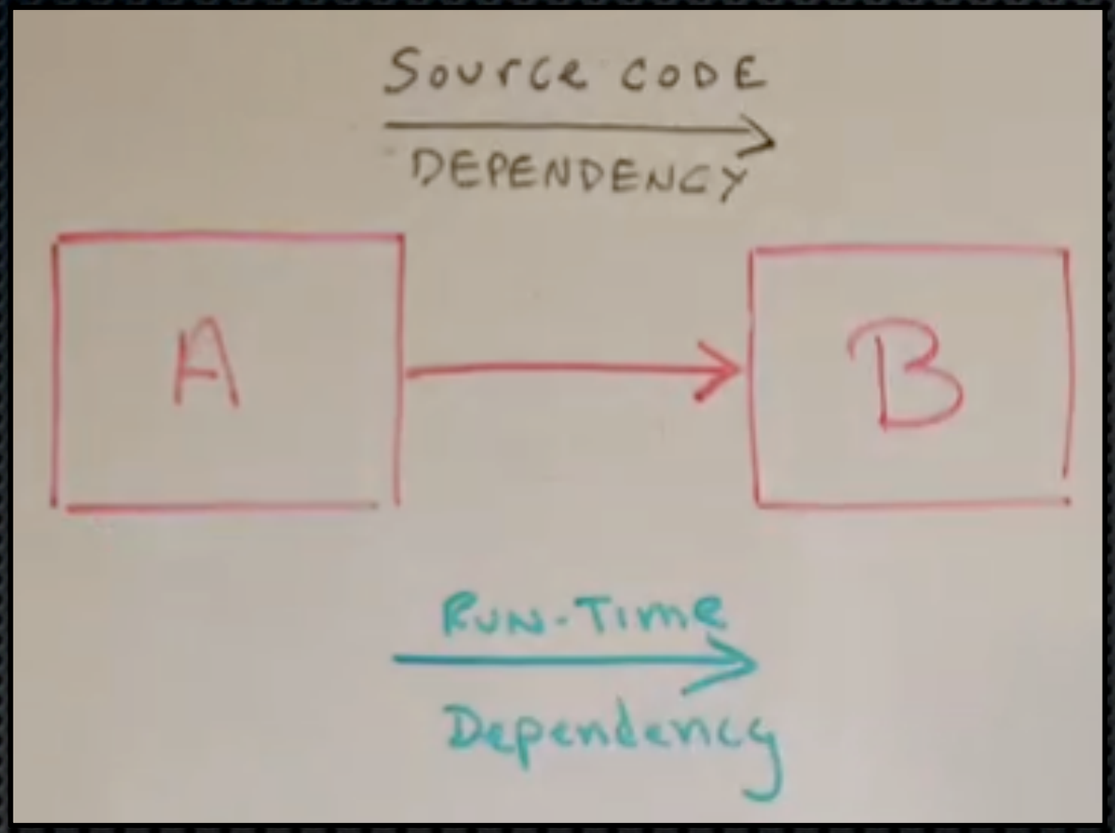
- 모듈 A와 B를 독립적으로 배포하고자 한다면 ?
  - 모듈 A가 모듈 B에 source code dependency를 가지면 독립적으로 배포될 수 없다

Switches and Cases



- 모듈 A와 B를 독립적으로 배포하고자 한다면 ?
  - 모듈 A가 모듈 B에 source code dependency를 가지면 독립적으로 배포될 수 없다
  - 독립적으로 컴파일될 수도 없다

Switches and Cases



- 모듈 A와 B를 독립적으로 배포하고자 한다면 ?
  - 모듈 A가 모듈 B에 source code dependency를 가지면 독립적으로 배포될 수 없다
  - 독립적으로 컴파일될 수도 없다
  - 모듈 B의 변경은 모듈 A의 재컴파일, 재배포를 유발

Switches and Cases

Switches and Cases

- 객체지향은 뭔가 대단한 것을 허용

Switches and Cases

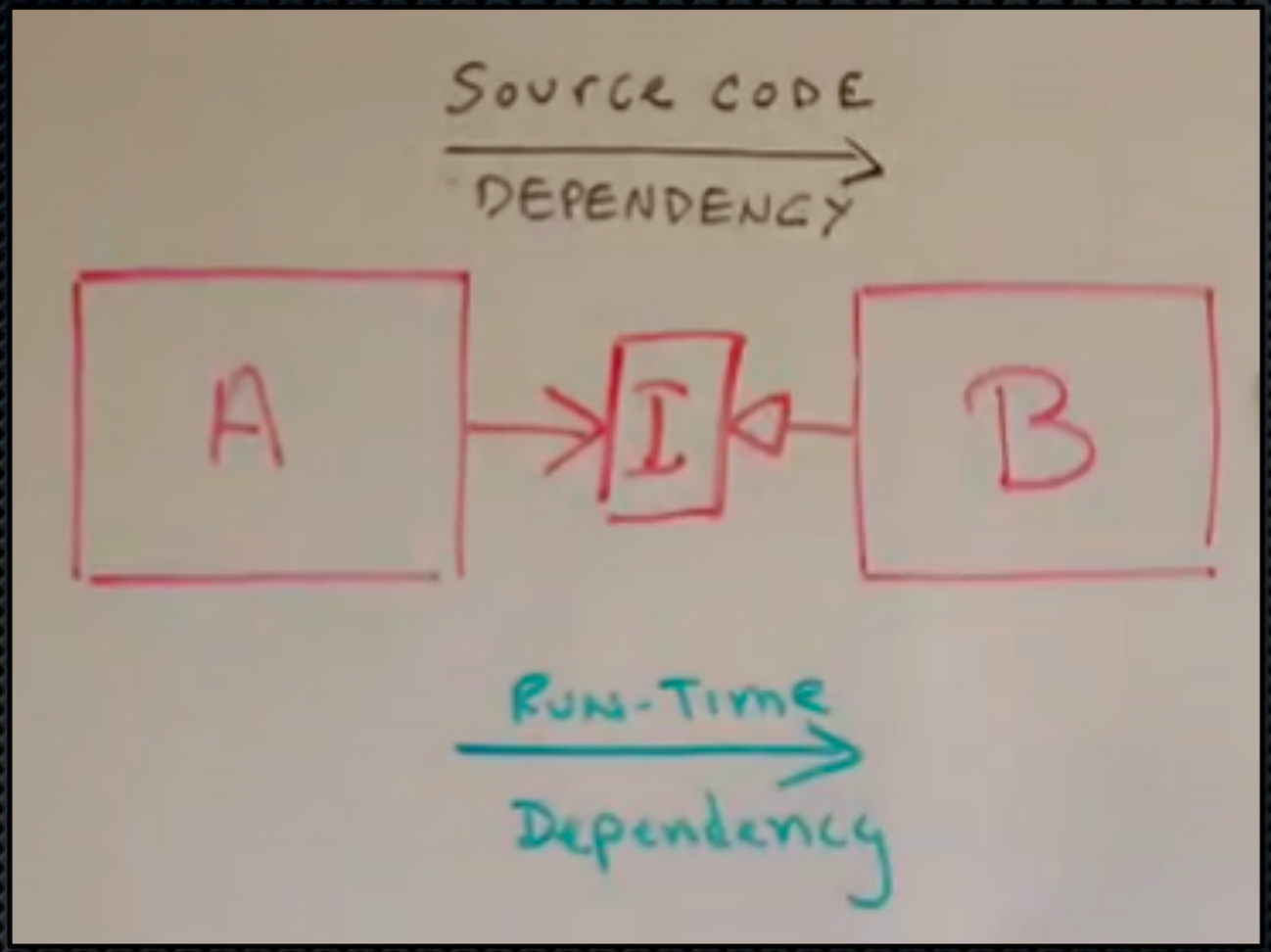
- 객체지향은 뭔가 대단한 것을 허용
  - RT 의존성은 그대로 둔채로

Switches and Cases

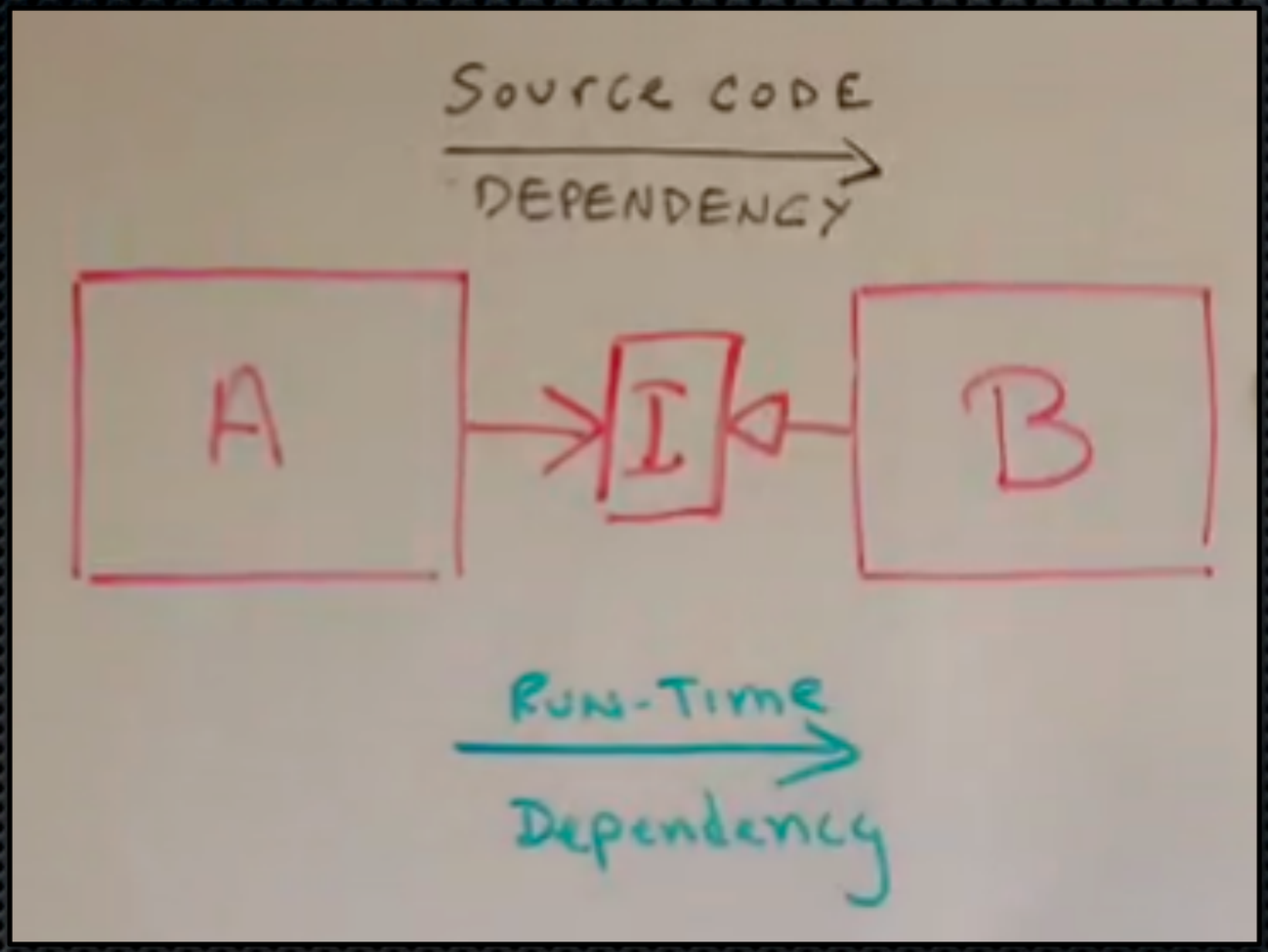
- 객체지향은 뭔가 대단한 것을 허용
  - RT 의존성은 그대로 둔채로
  - source code 의존성을 뒤집는다.

Switches and Cases

Switches and Cases

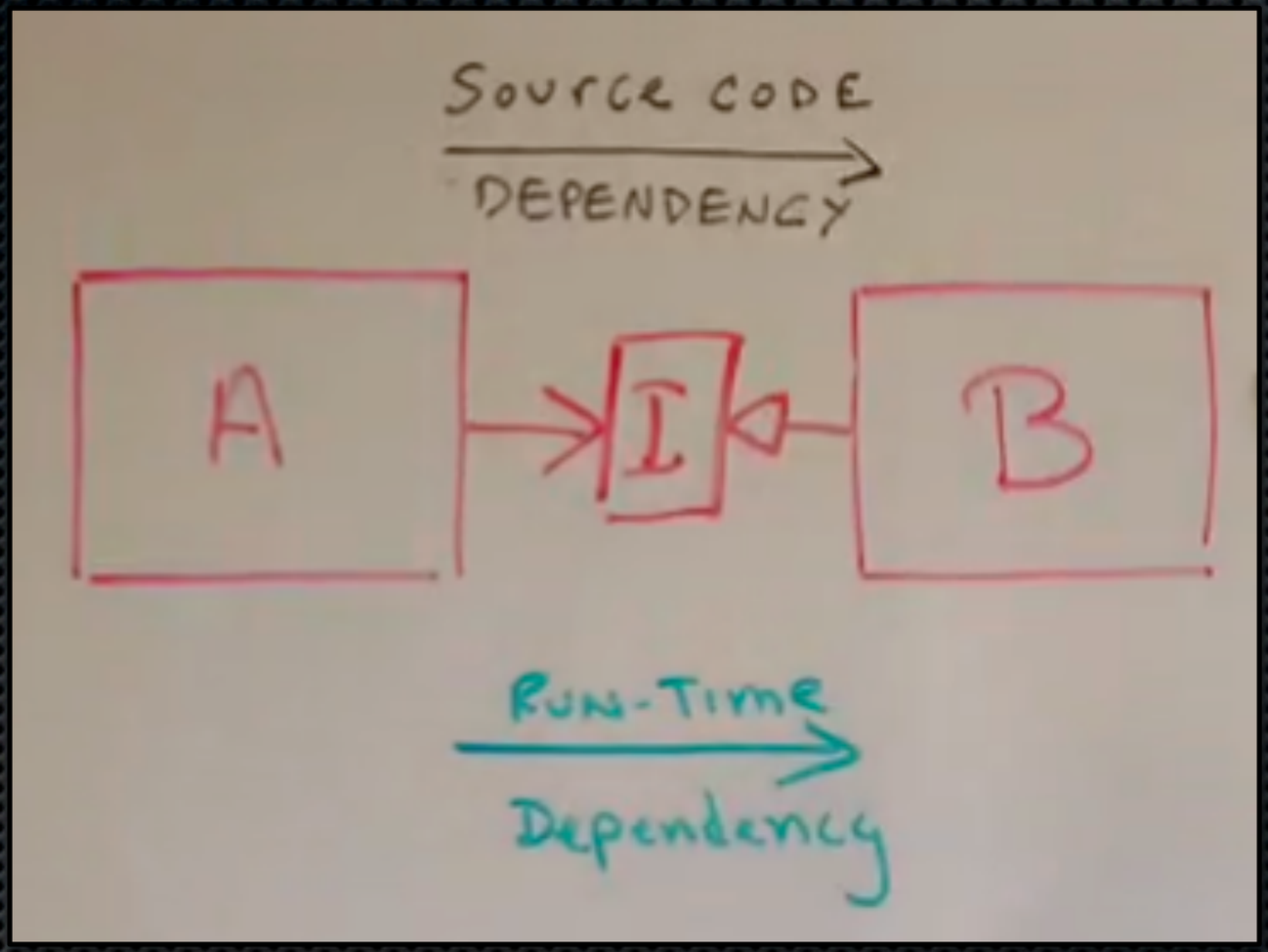


Switches and Cases



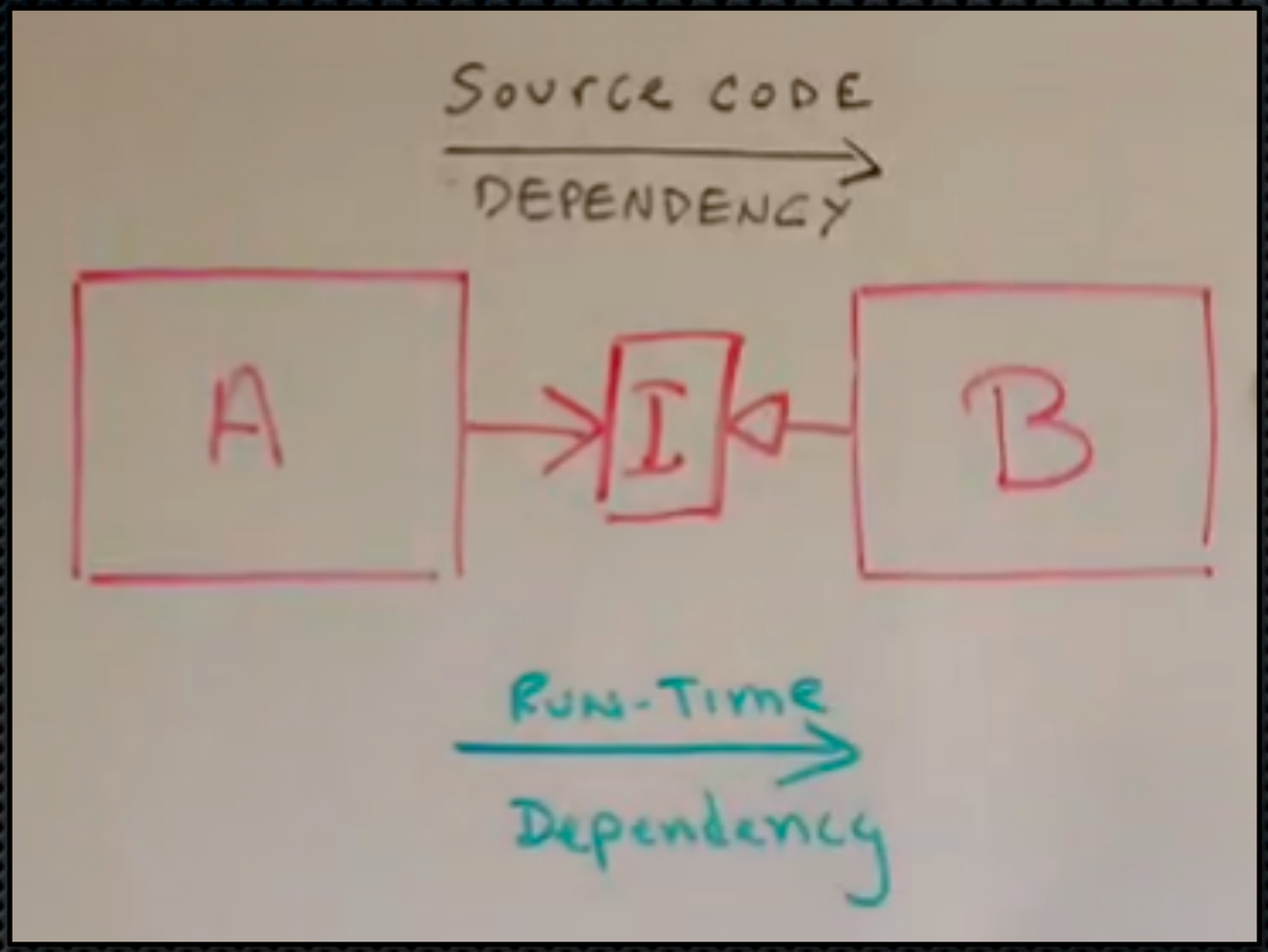
- 본래의 의존성을 제거

Switches and Cases



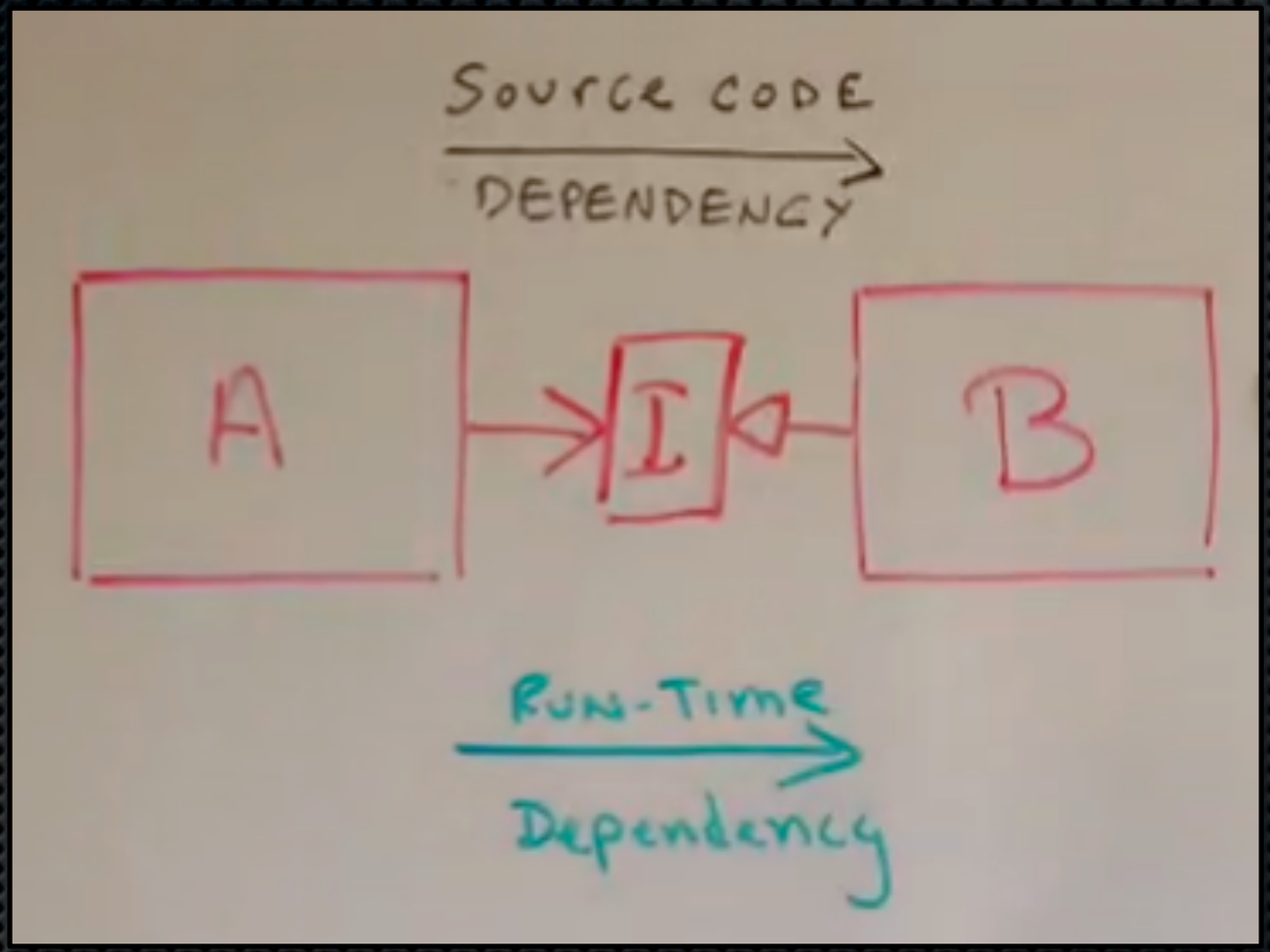
- 본래의 의존성을 제거
- polymorphic interface를 삽입

Switches and Cases



- 본래의 의존성을 제거
- polymorphic interface를 삽입
- 모듈 A는 인터페이스에 의존,

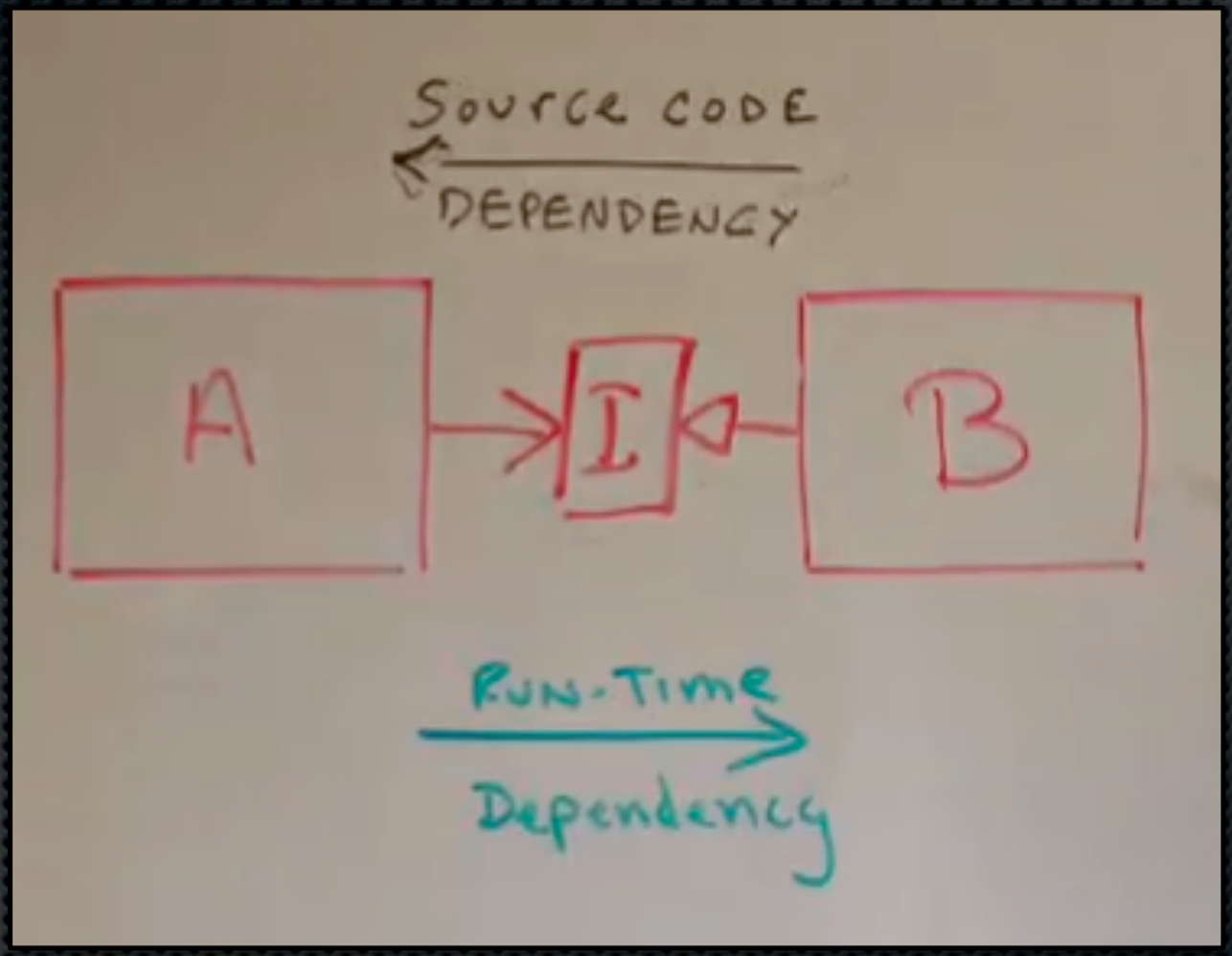
Switches and Cases



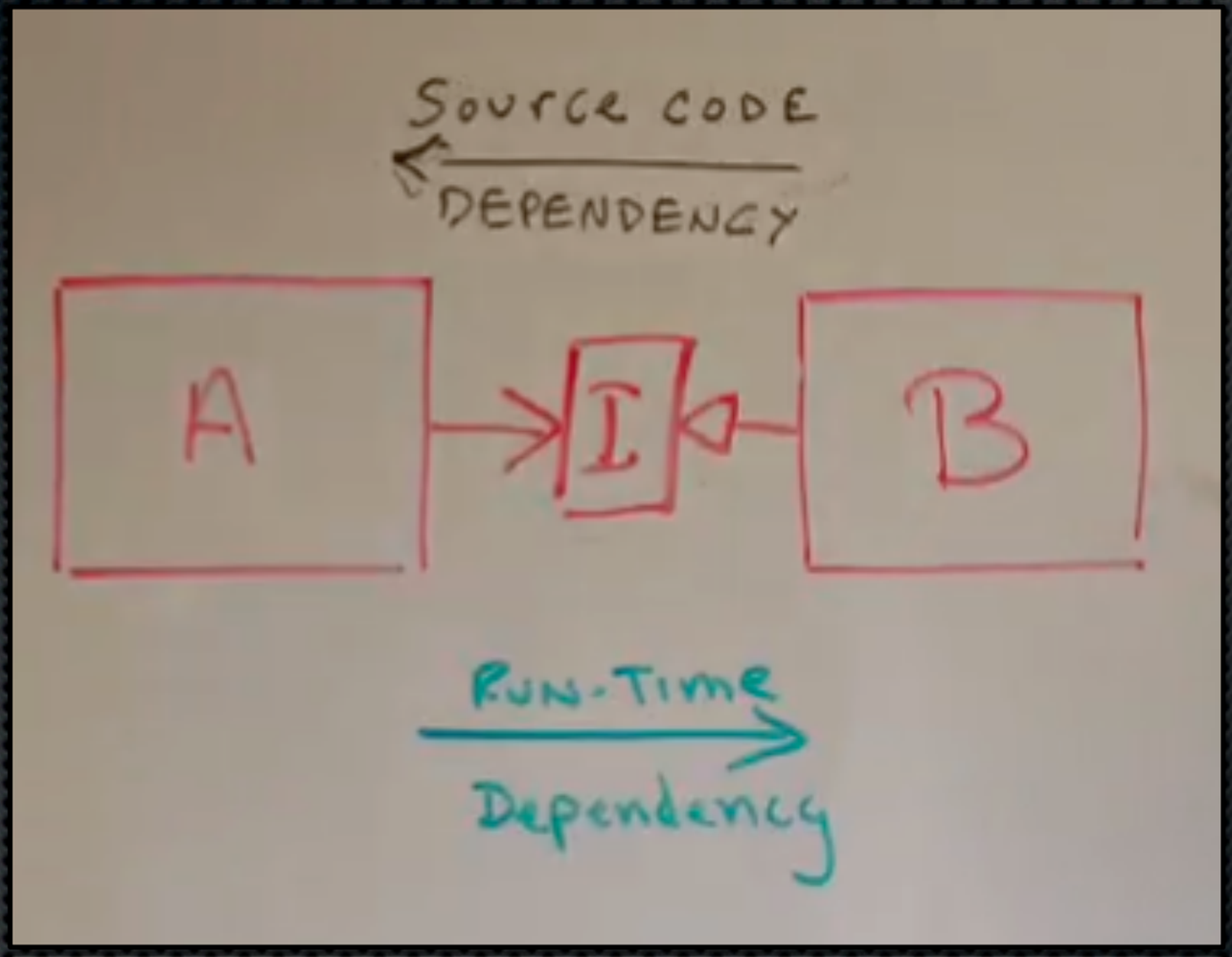
- 본래의 의존성을 제거
- polymorphic interface를 삽입
- 모듈 A는 인터페이스에 의존,
- 모듈 B는 인터페이스로부터 derive

Switches and Cases

Switches and Cases

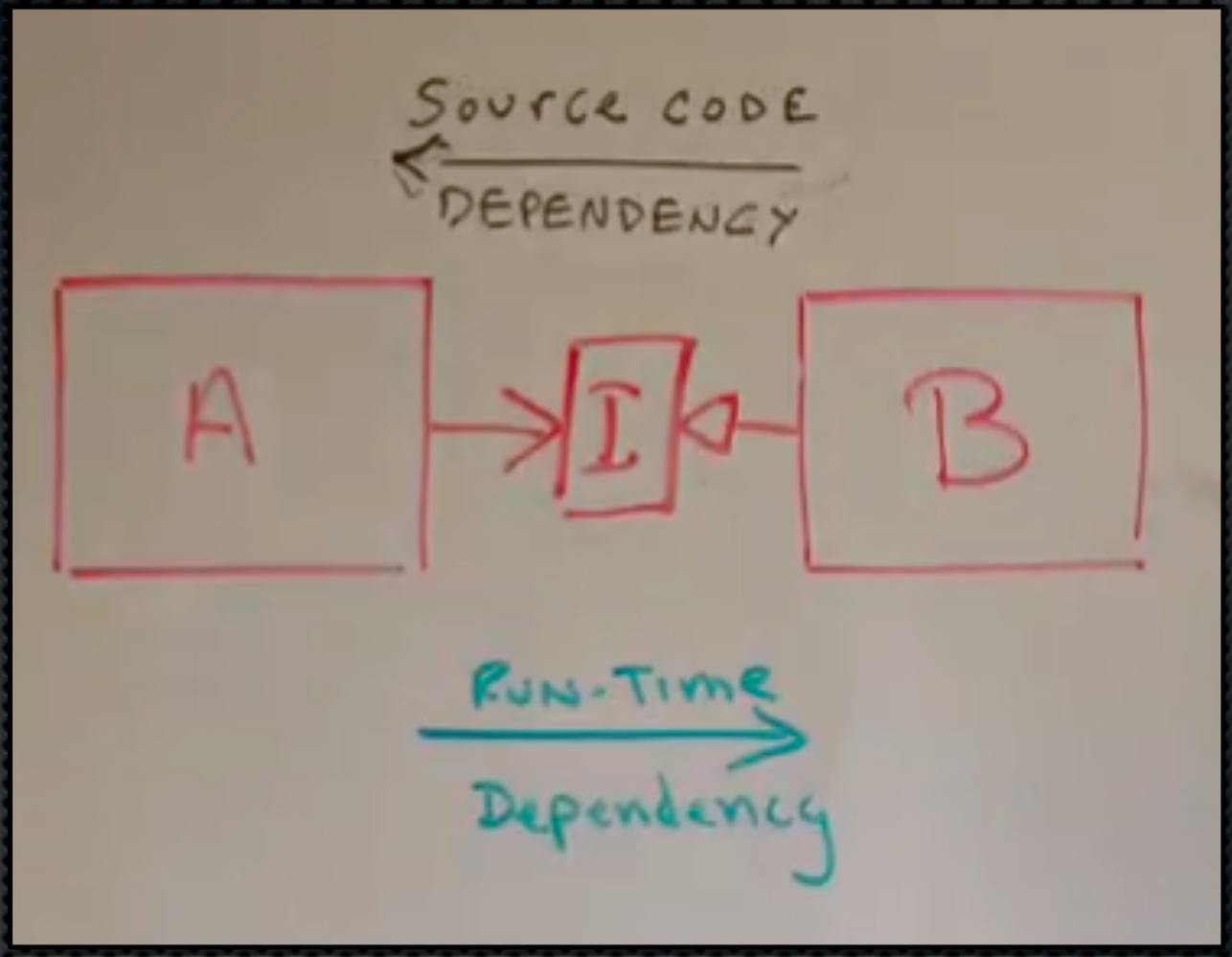


Switches and Cases



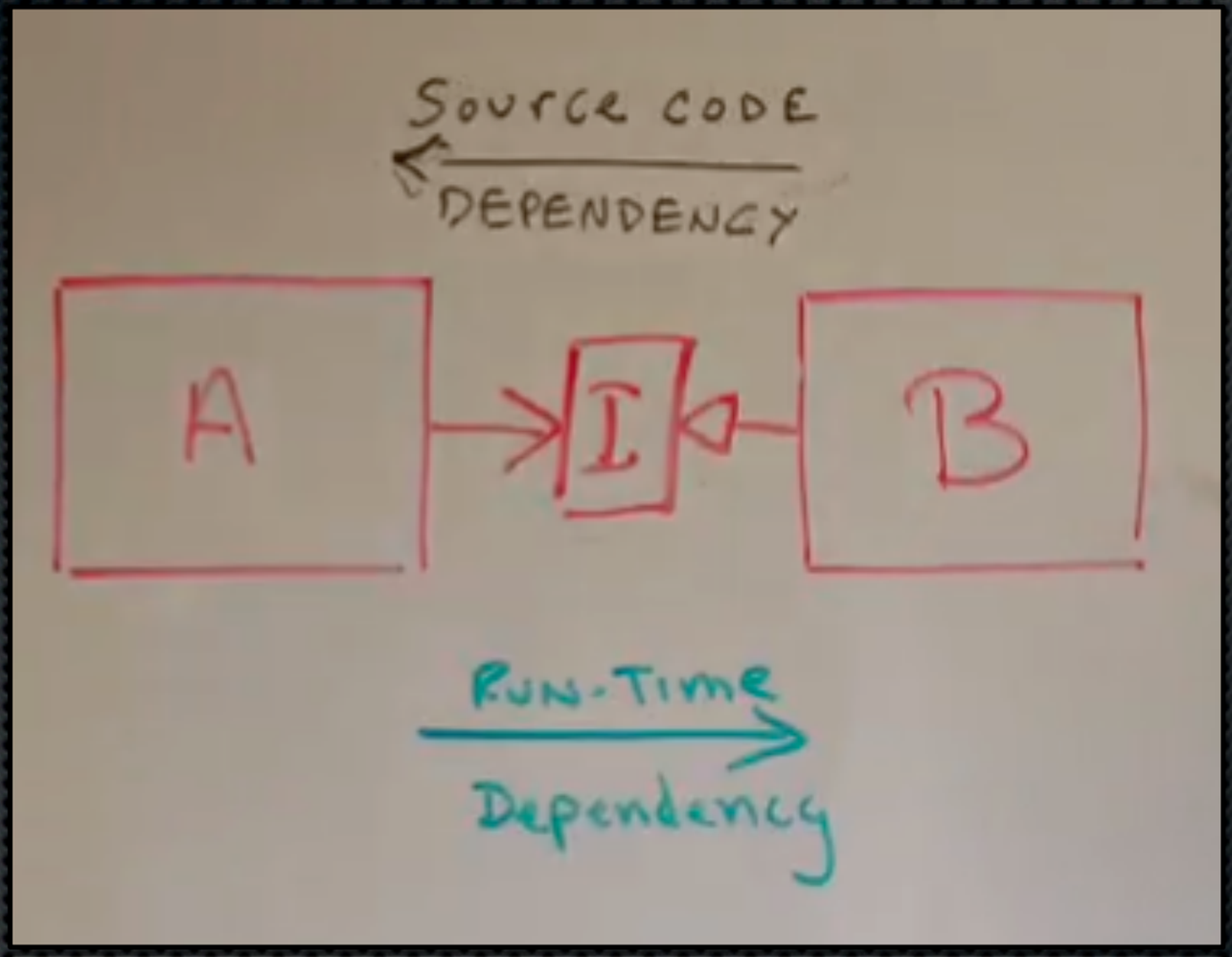
–모듈 B의 입장에서는 source code 의존성이 역전

Switches and Cases



- 모듈 B의 입장에서는 source code 의존성이 역전
- 이제 모듈 A,B는 독립적으로 배포 가능

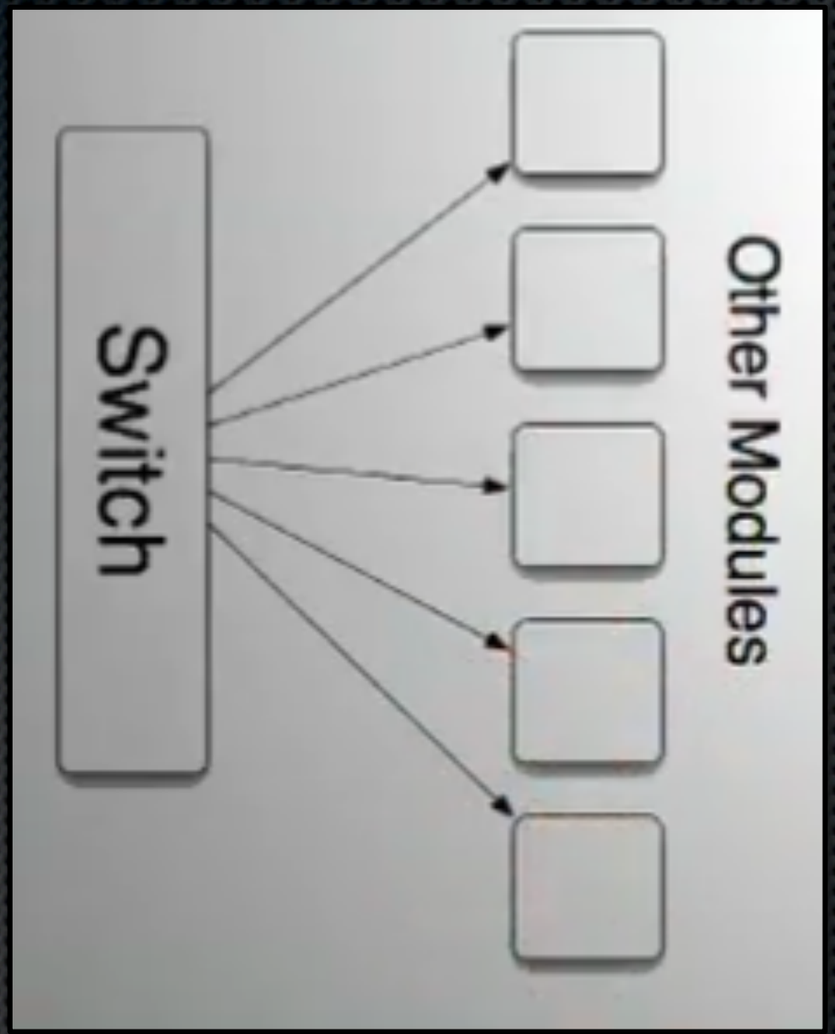
Switches and Cases



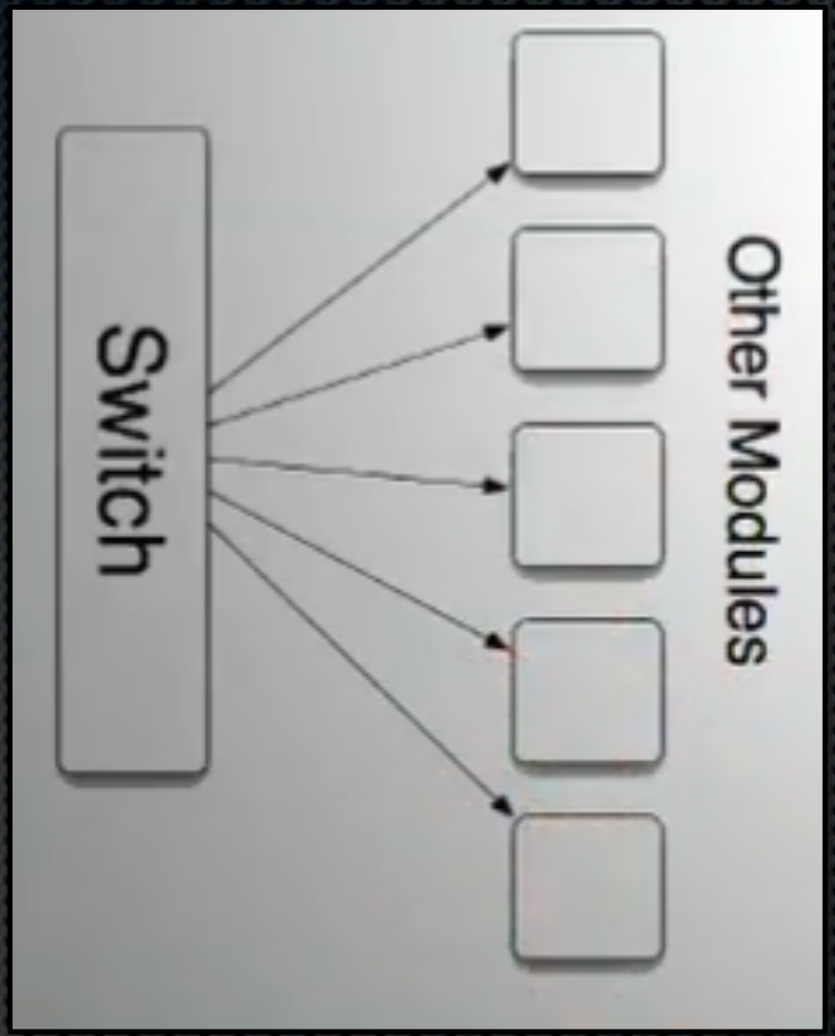
- 모듈 B의 입장에서는 source code 의존성이 역전
- 이제 모듈 A,B는 독립적으로 배포 가능
- Independent Deployability : one of OO의 강점

Switches and Cases

Switches and Cases

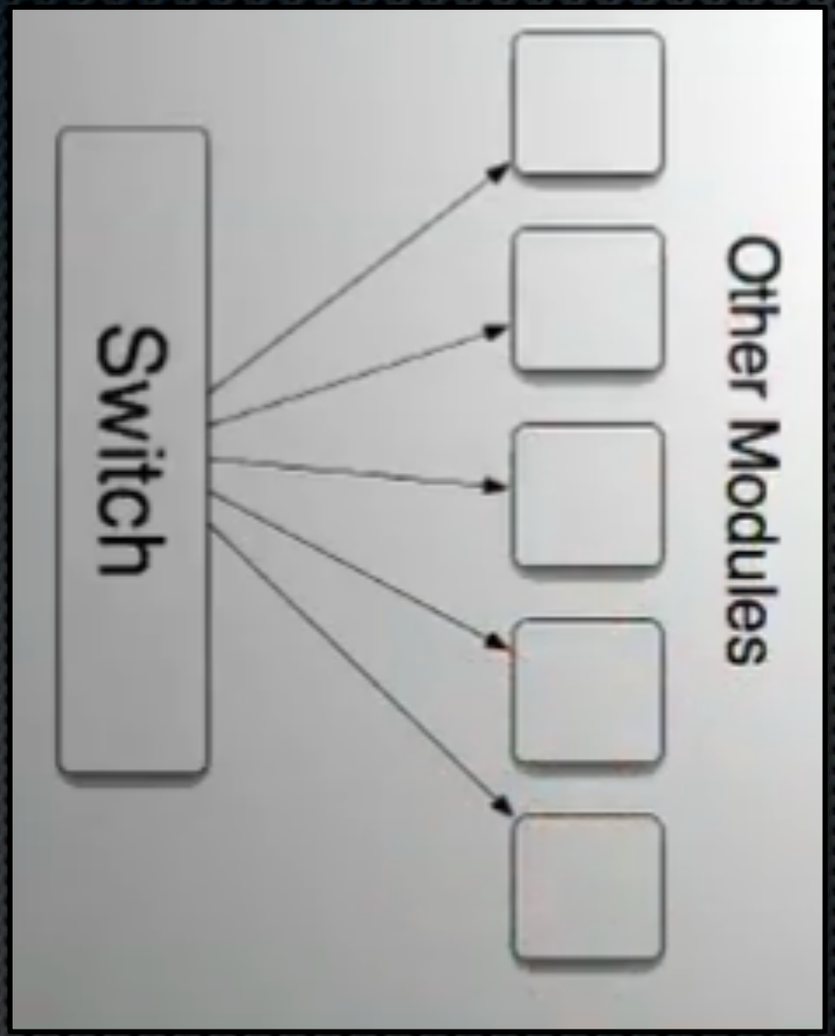


Switches and Cases



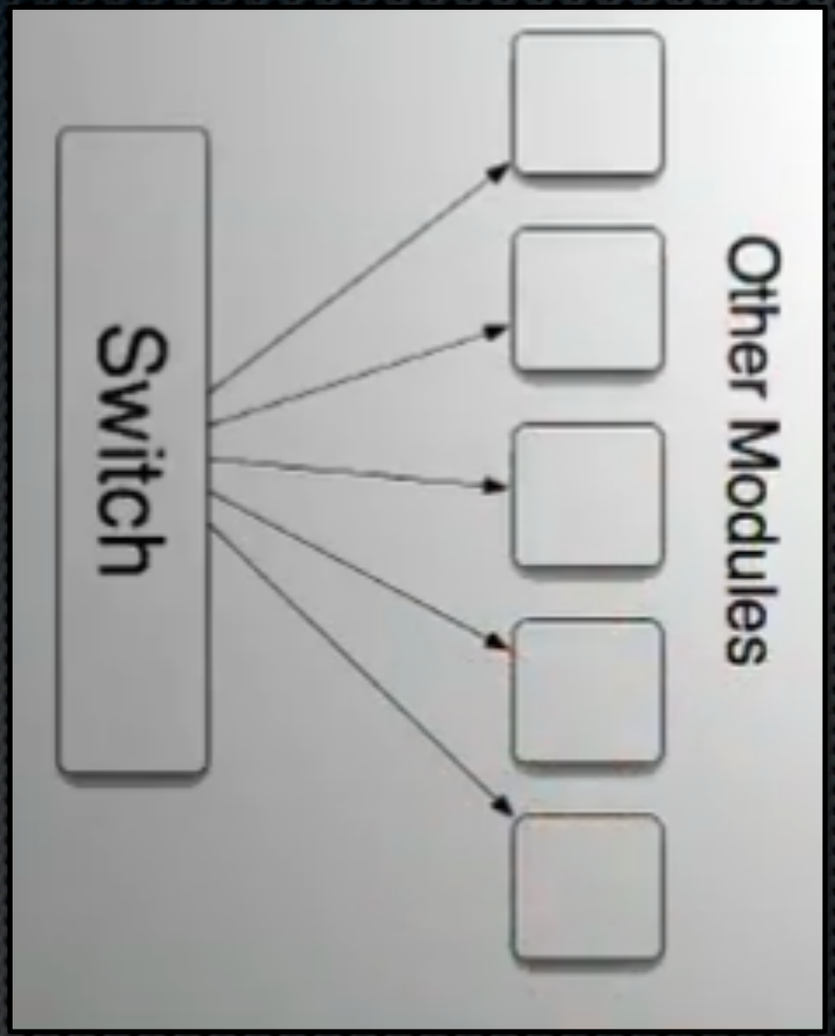
- switch 문장은 독립적 배포에 방해

Switches and Cases



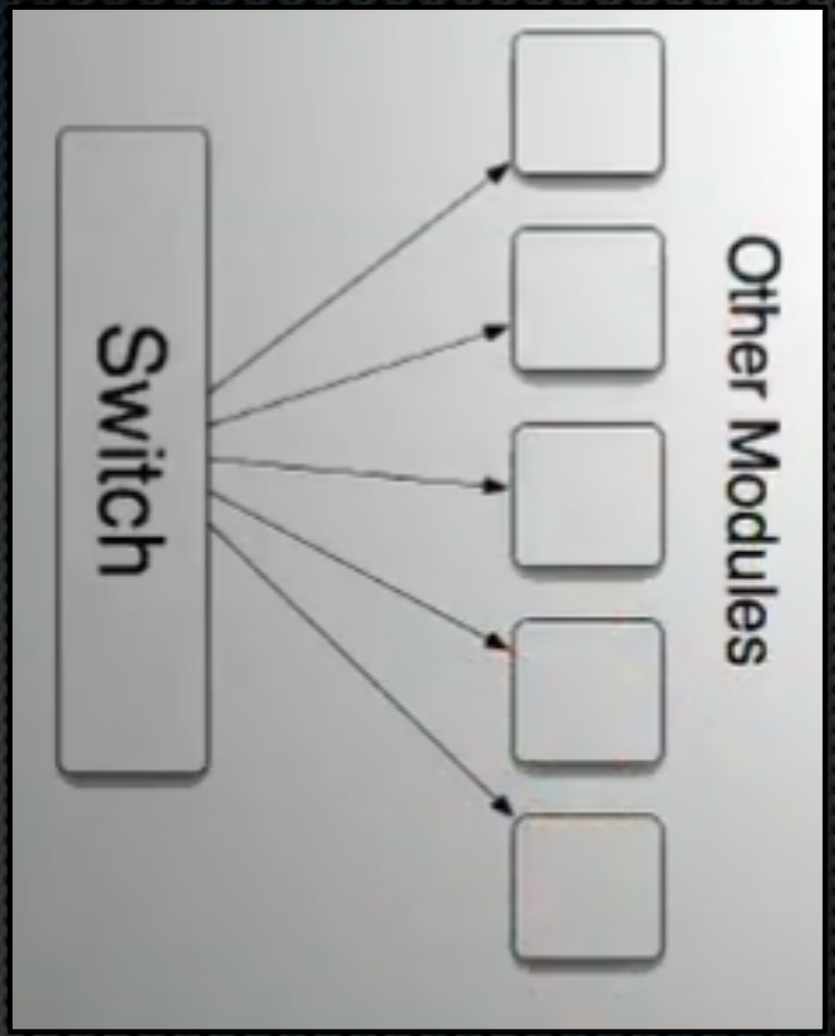
- switch 문장은 독립적 배포에 방해
  - 각 case 문장은 외부 모듈에 의존

Switches and Cases



- switch 문장은 독립적 배포에 방해
  - 각 case 문장은 외부 모듈에 의존
  - 다수의 다른 모듈에 대한 의존성 존재

Switches and Cases



- switch 문장은 독립적 배포에 방해
  - 각 case 문장은 외부 모듈에 의존
  - 다수의 다른 모듈에 대한 의존성 존재
  - fan-out problem

Switches and Cases

Switches and Cases

- switch 문장에서 source code 의존성은 flow of control과 방향이 같다.

Switches and Cases

- switch 문장에서 source code 의존성은 flow of control과 방향이 같다.
  - switch 문장은 모든 외부 모듈의 변경에 영향을 받음

Switches and Cases

- switch 문장에서 source code 의존성은 flow of control과 방향이 같다.
  - switch 문장은 모든 외부 모듈의 변경에 영향을 받음
  - 외부 모듈 중 하나라도 변경이 일어나면 switch 문장에 영향을 미치고,

Switches and Cases

- switch 문장에서 source code 의존성은 flow of control과 방향이 같다.
  - switch 문장은 모든 외부 모듈의 변경에 영향을 받음
  - 외부 모듈 중 하나라도 변경이 일어나면 switch 문장에 영향을 미치고,
  - switch에 의존하는 다른 모든 것들에도 영향을 미침

Switches and Cases

- switch 문장에서 source code 의존성은 flow of control과 방향이 같다.
  - switch 문장은 모든 외부 모듈의 변경에 영향을 받음
  - 외부 모듈 중 하나라도 변경이 일어나면 switch 문장에 영향을 미치고,
  - switch에 의존하는 다른 모든 것들에도 영향을 미침
  - switch 문장은 독립적 배포를 불가능하게 하는 많은 의존성을 만듦

Switches and Cases

Switches and Cases

- switch 문장 제거

Switches and Cases

- switch 문장 제거
  - invert switch to polymorphic dispatch

Switches and Cases

- switch 문장 제거
  - invert switch to polymorphic dispatch
  - move switch stmt. to places that can't do any harm

Switches and Cases

- switch 문장 제거
  - invert switch to polymorphic dispatch
  - move switch stmt. to places that can't do any harm
- switch 문장의 인자(타입 코드)를 abstract base 클래스로 치환

Switches and Cases

- switch 문장 제거
  - invert switch to polymorphic dispatch
  - move switch stmt. to places that can't do any harm
- switch 문장의 인자(타입 코드)를 abstract base 클래스로 치환
  - case 문장에서 수행할 코드에 대응하는 메소드를 갖는.

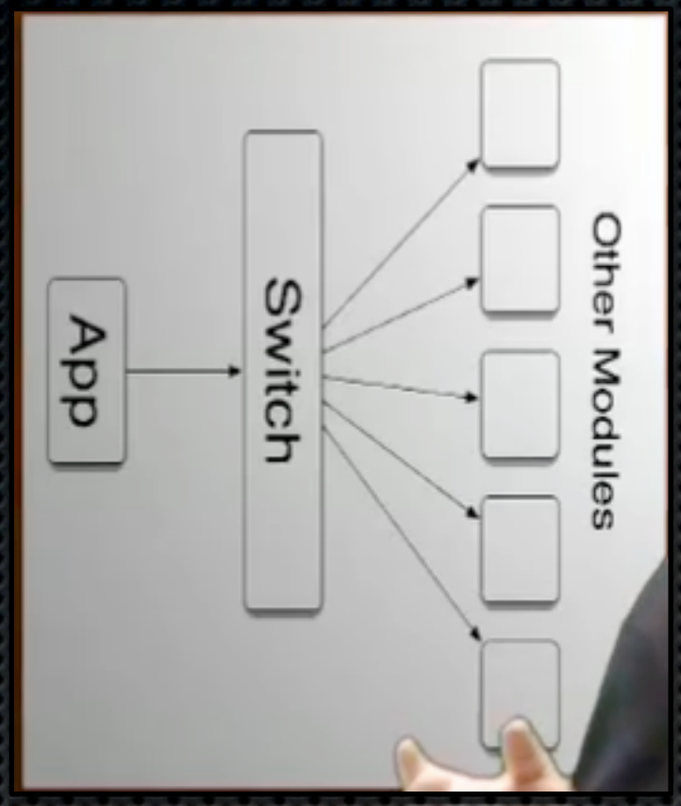
Switches and Cases

- 실습

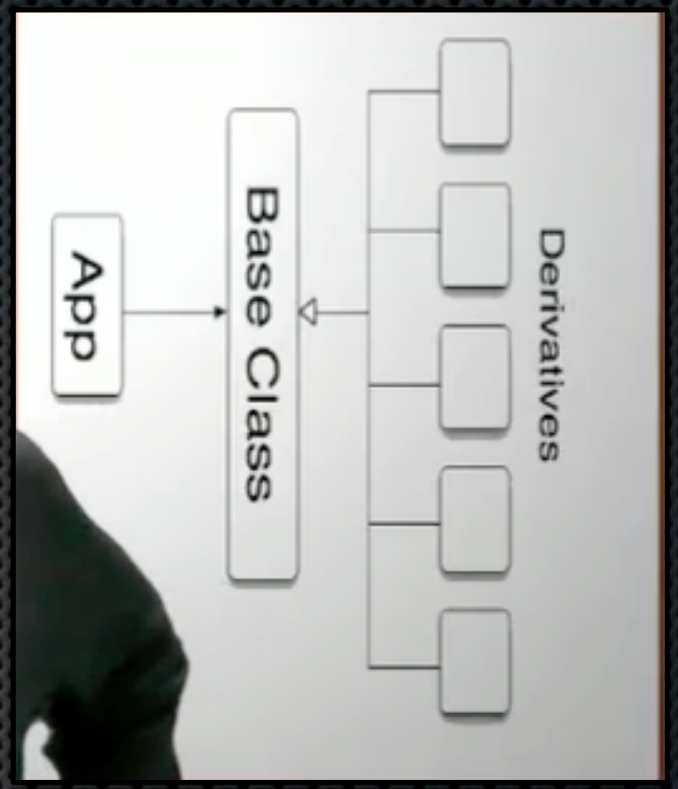
https://github.com/msbaek/videostore

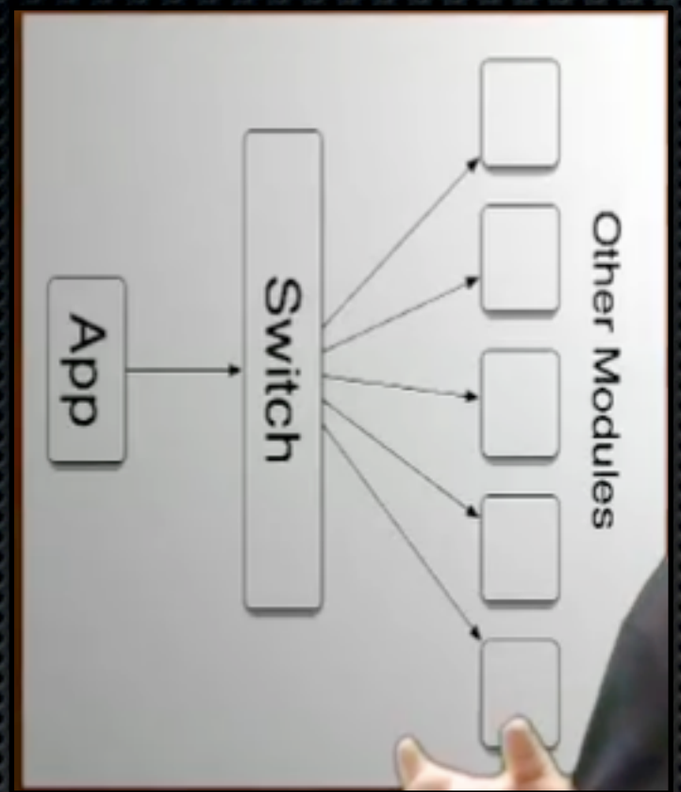
Switches and Cases

Switches and Cases

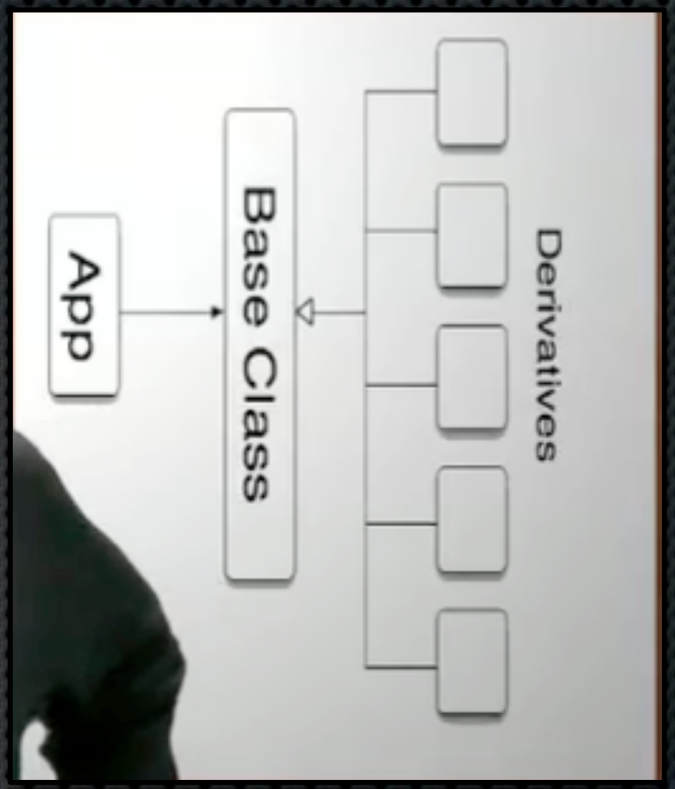


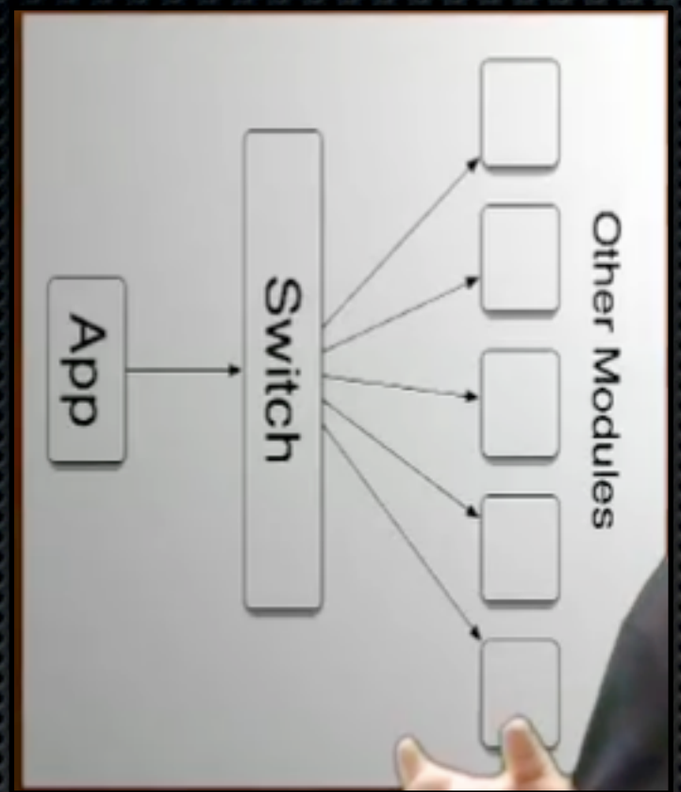
Switches and Cases





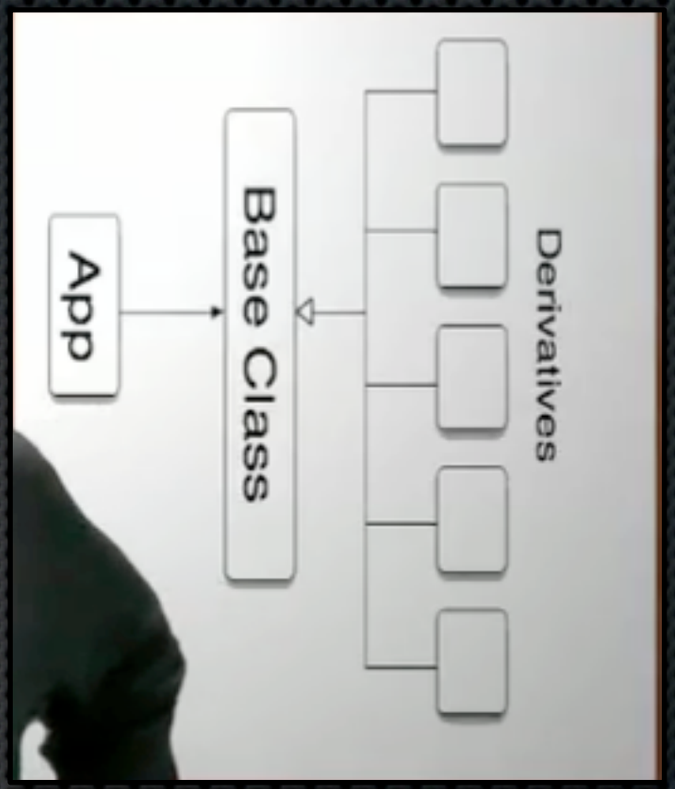
Switches and Cases

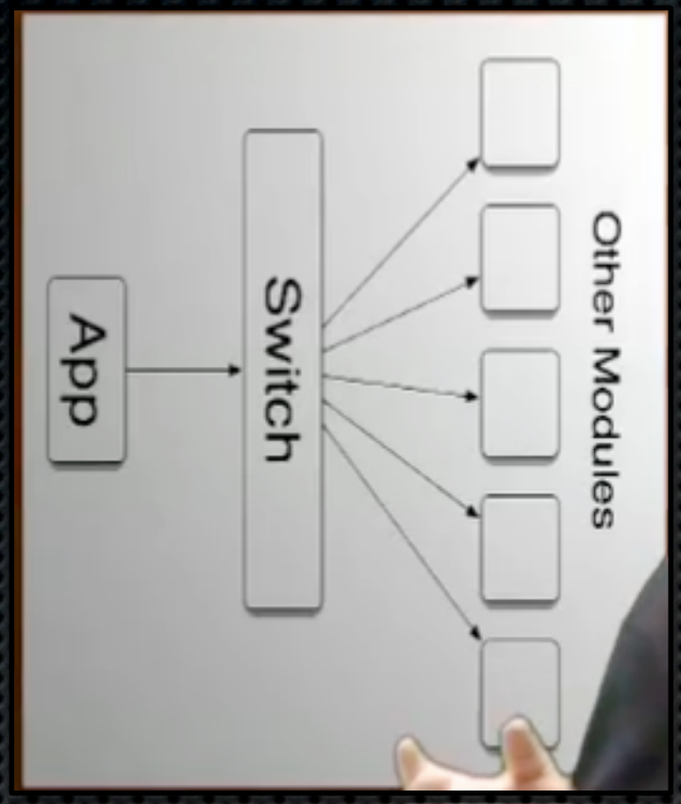




- RT 의존성은 그대로 이나 소스 코드 의존성이 역전됨

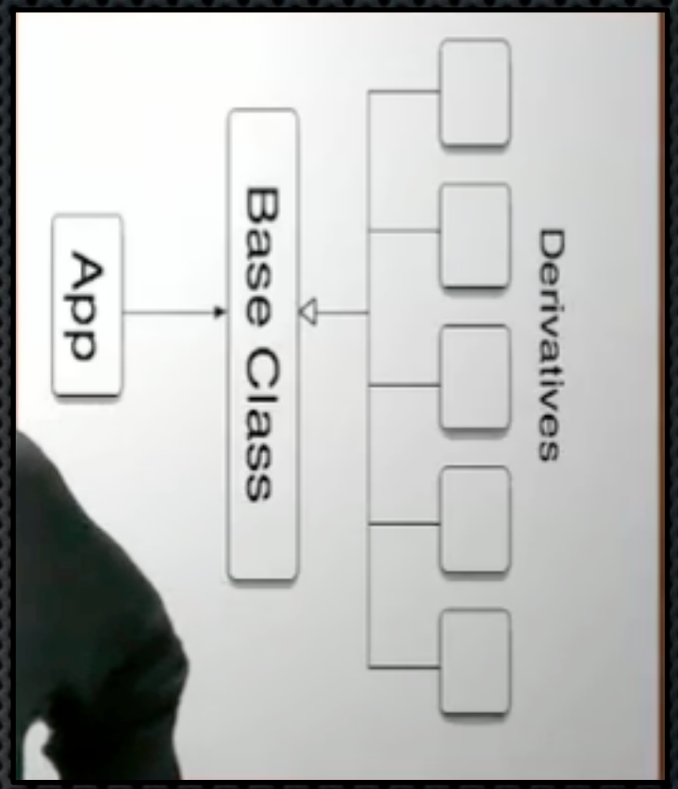
Switches and Cases

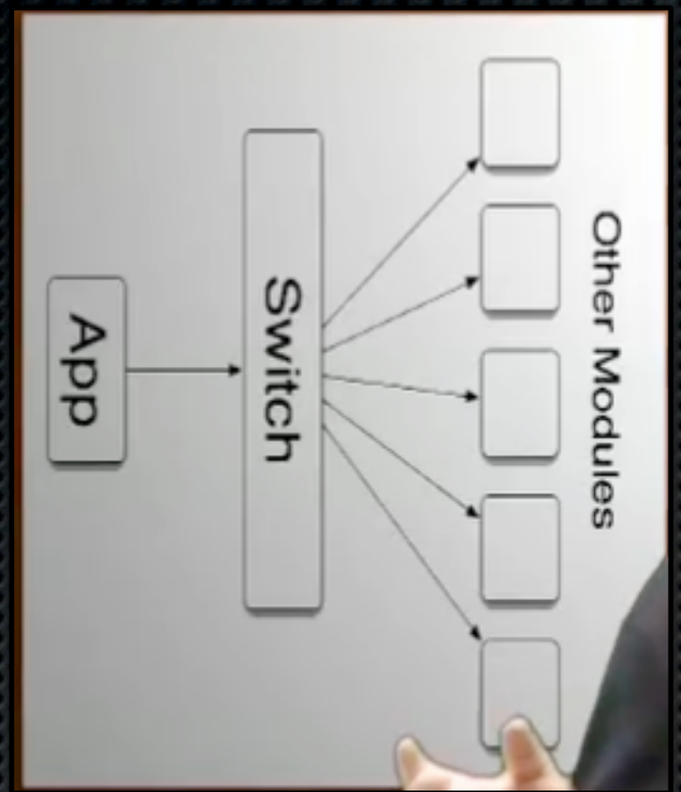




- RT 의존성은 그대로 이나 소스 코드 의존성이 역전됨
- 이제 신경 쓸 일은 언제/어떻게 인스턴스들을 생성하는가이다.

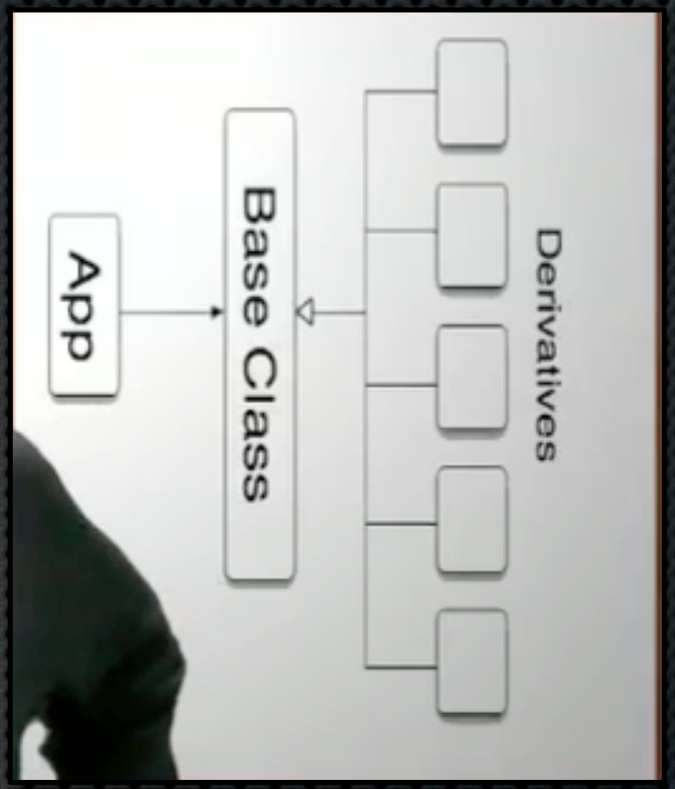
Switches and Cases

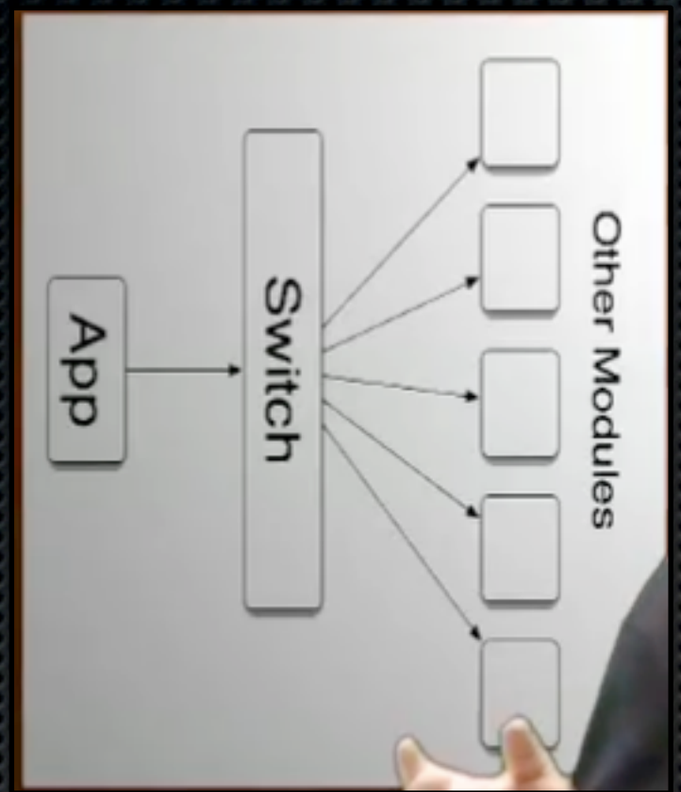




- RT 의존성은 그대로 이나 소스 코드 의존성이 역전됨
- 이제 신경 쓸 일은 언제/어떻게 인스턴스들을 생성하는가이다.
- 대개 팩토리에서 이런 작업을 수행.

Switches and Cases

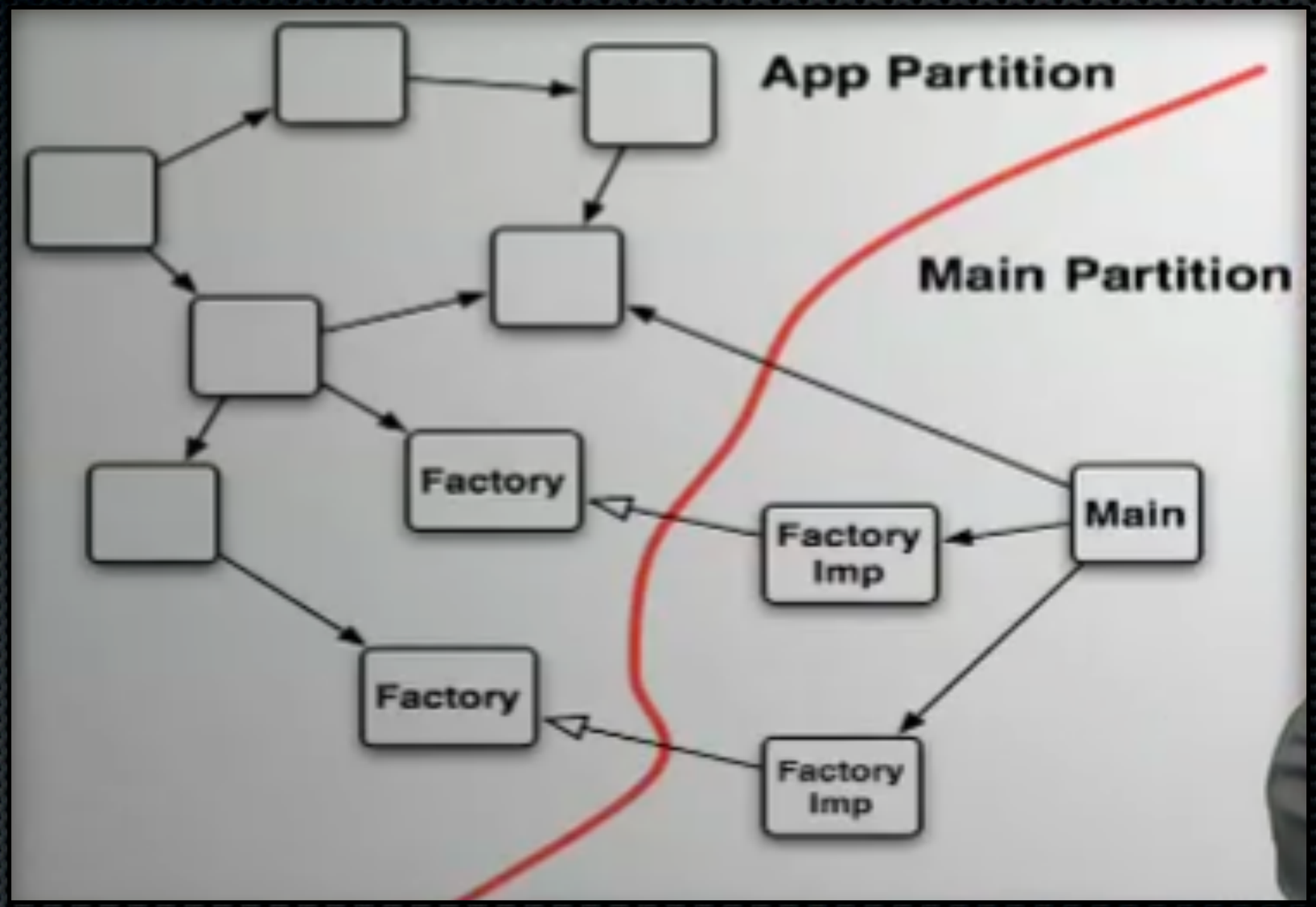




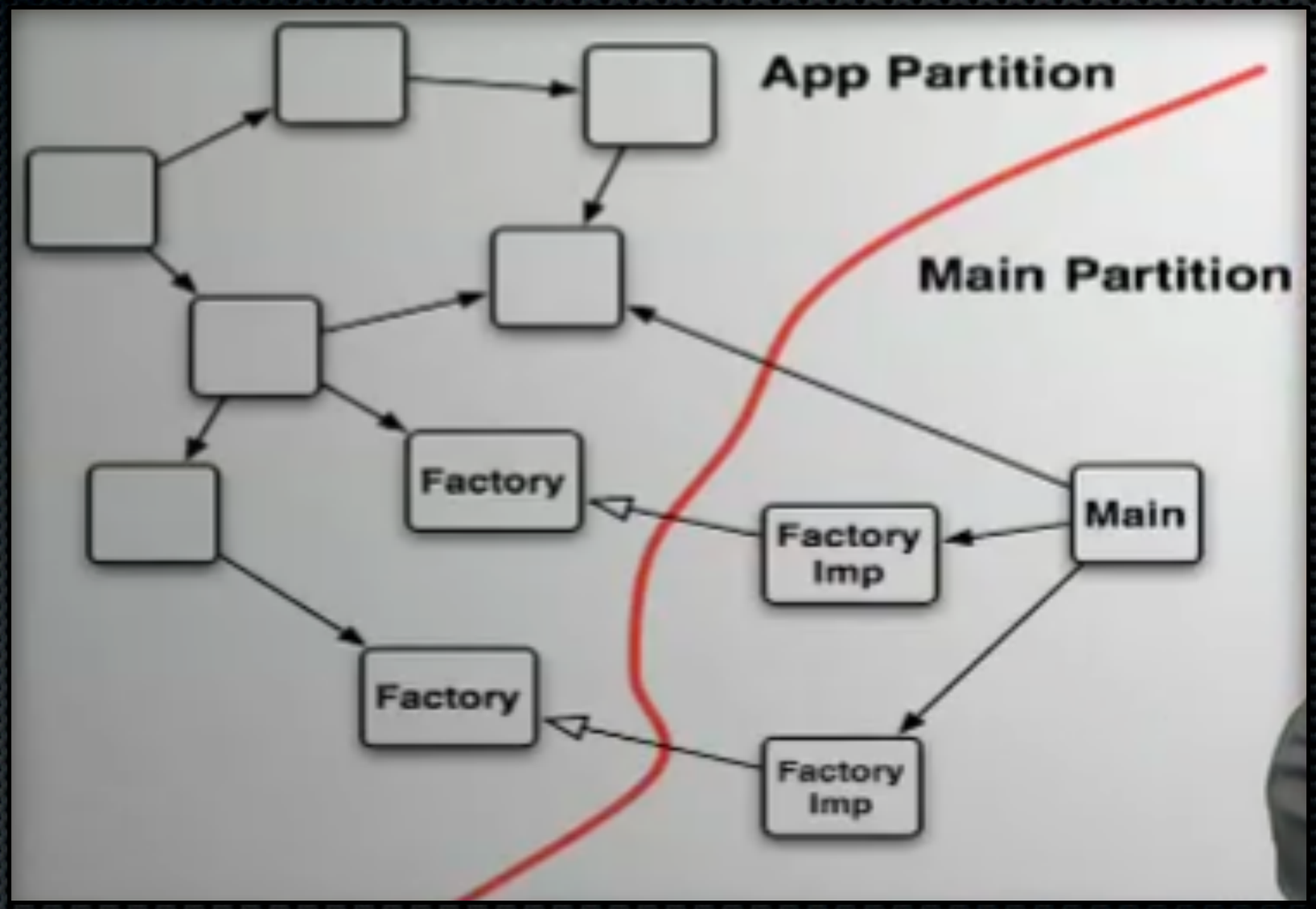
- RT 의존성은 그대로 이나 소스 코드 의존성이 역전됨
- 이제 신경 쓸 일은 언제/어떻게 인스턴스들을 생성하는가이다.
- 대개 팩토리에서 이런 작업을 수행.
- 이게 main이라는 주요한 토픽을 가져온다.

Switches and Cases

Switches and Cases



Switches and Cases



- 항상 모듈 다이어그램에서 App 파티션과와 Main 파티션 사이에 선을 그을 수 있어야

Switches and Cases

Switches and Cases

- 2개의 Partition

Switches and Cases

- 2개의 Partition
- Application Partition

Switches and Cases

- 2개의 Partition
- Application Partition
  - 대개의 어플리케이션 코드가 존재하는 곳

Switches and Cases

- 2개의 Partition
- Application Partition
  - 대개의 어플리케이션 코드가 존재하는 곳
- Main Partition

Switches and Cases

- 2개의 Partition
- Application Partition
  - 대개의 어플리케이션 코드가 존재하는 곳
- Main Partition
  - 하위 레벨 Details

Switches and Cases

- 2개의 Partition
- Application Partition
  - 대개의 어플리케이션 코드가 존재하는 곳
- Main Partition
  - 하위 레벨 Details
  - factory, configuration data, main program

Switches and Cases

- 2개의 Partition
- Application Partition
  - 대개의 어플리케이션 코드가 존재하는 곳
- Main Partition
  - 하위 레벨 Details
  - factory, configuration data, main program
  - should be kept small

Switches and Cases

- 2개의 Partition
- Application Partition
  - 대개의 어플리케이션 코드가 존재하는 곳
- Main Partition
  - 하위 레벨 Details
  - factory, configuration data, main program
  - should be kept small
- 한방향의 의존성

Switches and Cases

- 2개의 Partition
- Application Partition
  - 대개의 어플리케이션 코드가 존재하는 곳
- Main Partition
  - 하위 레벨 Details
  - factory, configuration data, main program
  - should be kept small
- 한방향의 의존성
  - from Main to App

Switches and Cases

- 2개의 Partition
- Application Partition
  - 대개의 어플리케이션 코드가 존재하는 곳
- Main Partition
  - 하위 레벨 Details
  - factory, configuration data, main program
  - should be kept small
- 한방향의 의존성
  - from Main to App
- Main Partition은 App Partition의 plugin

Switches and Cases

- 2개의 Partition
- Application Partition
  - 대개의 어플리케이션 코드가 존재하는 곳
- Main Partition
  - 하위 레벨 Details
  - factory, configuration data, main program
  - should be kept small
- 한방향의 의존성
  - from Main to App
- Main Partition은 App Partition의 plugin
- Dependency Injection

Temporal Coupling

Temporal Coupling

- 함수들이 순서를 지켜며 호출되어야 한다.

Temporal Coupling

- 함수들이 순서를 지켜며 호출되어야 한다.
- temporal coupling을 지키지 않아서 오류를 겪은 적이 있을 것

Temporal Coupling

- 함수들이 순서를 지켜며 호출되어야 한다.
- temporal coupling을 지키지 않아서 오류를 겪은 적이 있을 것
- open/close처럼 쉽지만은 않다.

Temporal Coupling

- 함수들이 순서를 지켜며 호출되어야 한다.
- temporal coupling을 지키지 않아서 오류를 겪은 적이 있을 것
- open/close처럼 쉽지만은 않다.
- 대부분의 temporal coupling은 감춰져있다.

Temporal Coupling

Temporal Coupling

- Passing a Block

Temporal Coupling

- Passing a Block

```
1 // file should be opened before processing
2 fileCommand.process(file);
3 // file should be closed after processing
```

Temporal Coupling

- Passing a Block

```
1 // file should be opened before processing
2 fileCommand.process(file);
3 // file should be closed after processing
```

```
7 fileCommandTemplate.process(myfile, new FileCommand() {
8     public void process(File f) {
9         // file processing codes here
10    }
11 });
12
13 class FileCommandTemplate {
14    public void process(File file, FileCommand command) {
15        file.open();
16        command.process(file);
17        file.close();
18    }
19 }
```

Command Query Separation

Command Query Separation

- side effect를 관리하는 좋은 방법

Command Query Separation

- side effect를 관리하는 좋은 방법
- command

Command Query Separation

- side effect를 관리하는 좋은 방법
- command
  - 시스템의 상태 변경 가능

Command Query Separation

- side effect를 관리하는 좋은 방법
- command
  - 시스템의 상태 변경 가능
  - side effect를 갖는다.

Command Query Separation

- side effect를 관리하는 좋은 방법
- command
  - 시스템의 상태 변경 가능
  - side effect를 갖는다.
  - 아무것도 반환하지 않는다.

Command Query Separation

- side effect를 관리하는 좋은 방법
- command
  - 시스템의 상태 변경 가능
  - side effect를 갖는다.
  - 아무것도 반환하지 않는다.
- query

Command Query Separation

- side effect를 관리하는 좋은 방법
- command
  - 시스템의 상태 변경 가능
  - side effect를 갖는다.
  - 아무것도 반환하지 않는다.
- query
  - side effect가 없다.

Command Query Separation

- side effect를 관리하는 좋은 방법
- command
  - 시스템의 상태 변경 가능
  - side effect를 갖는다.
  - 아무것도 반환하지 않는다.
- query
  - side effect가 없다.
  - 계산값이나 시스템의 상태를 반환

Command Query Separation

- side effect를 관리하는 좋은 방법
- command
  - 시스템의 상태 변경 가능
  - side effect를 갖는다.
  - 아무것도 반환하지 않는다.
- query
  - side effect가 없다.
  - 계산값이나 시스템의 상태를 반환
- CQS 정의

Command Query Separation

- side effect를 관리하는 좋은 방법
- command
  - 시스템의 상태 변경 가능
  - side effect를 갖는다.
  - 아무것도 반환하지 않는다.
- query
  - side effect가 없다.
  - 계산값이나 시스템의 상태를 반환
- CQS 정의
  - 상태를 변경하는 함수는 값을 반환하면 안된다.

Command Query Separation

- side effect를 관리하는 좋은 방법
- command
  - 시스템의 상태 변경 가능
  - side effect를 갖는다.
  - 아무것도 반환하지 않는다.
- query
  - side effect가 없다.
  - 계산값이나 시스템의 상태를 반환
- CQS 정의
  - 상태를 변경하는 함수는 값을 반환하면 안된다.
  - 값을 반환하는 함수는 상태를 변경하면 안된다.

Command Query Separation

Command Query Separation

```
User u = authorizer.login(userName, password);
```

Command Query Separation

```
User u = authorizer.login(userName, password);
```

- user를 사용하기 위해서는 항상 login을 해야만 함.

Command Query Separation

```
User u = authorizer.login(userName, password);
```

- user를 사용하기 위해서는 항상 login을 해야만 함.
- login을 하면 원치 않아도 user 정보를 읽어야만 함.

Command Query Separation

```
2 User u = authorizer.login(userName, password);
```

- user를 사용하기 위해서는 항상 login을 해야만 함.
- login을 하면 원치 않아도 user 정보를 읽어야만 함.

```
7 authorizer.login(userName, password);
8 User u = authorizer.getUser(userName);
```

Command Query Separation

```
2 User u = authorizer.login(userName, password);
```

- user를 사용하기 위해서는 항상 login을 해야만 함.
- login을 하면 원치 않아도 user 정보를 읽어야만 함.

```
7 authorizer.login(userName, password);
8 User u = authorizer.getUser(userName);
```

- 당신 코드의 독자들을 혼란스럽게 하지 말라.

Command Query Separation

```
2 User u = authorizer.login(userName, password);
```

- user를 사용하기 위해서는 항상 login을 해야만 함.
- login을 하면 원치 않아도 user 정보를 읽어야만 함.

```
7 authorizer.login(userName, password);
8 User u = authorizer.getUser(userName);
```

- 당신 코드의 독자들을 혼란스럽게 하지 말라.
  - 값을 반환하는 함수는 상태를 변경하면 안된다.

Command Query Separation

```
2 User u = authorizer.login(userName, password);
```

- user를 사용하기 위해서는 항상 login을 해야만 함.
- login을 하면 원치 않아도 user 정보를 읽어야만 함.

```
7 authorizer.login(userName, password);
8 User u = authorizer.getUser(userName);
```

- 당신 코드의 독자들을 혼란스럽게 하지 말라.
  - 값을 반환하는 함수는 상태를 변경하면 안된다.
  - 상태를 변경하는 함수는 exception을 발생시킬 수는 있지만, 값을 반환할 수는 없다.

Tell, Don't Ask

Tell, Don't Ask

```
if(user.isLoggedIn())
    user.execute(command);
else
    authenticator.promptLogin();
```

Tell, Don't Ask

```
if(user.isLoggedIn())
    user.execute(command);
else
    authenticator.promptLogin();
```

```
try {
    user.execute(command);
}
catch(User.NotLoggedIn e) {
    annuciator.promptLogin();
}
```

Tell, Don't Ask

```
1 if(user.isLoggedIn())
2     user.execute(command);
3 else
4     authenticator.promptLogin();
```

```
6 try {
7     user.execute(command);
8 }
9 catch(User.NotLoggedIn e) {
10    annuciator.promptLogin();
11 }
```

```
13 user.execute(command, annuciator);
```

Tell, Don't Ask

Tell, Don't Ask

- 결국 로그인되었는지 아닌지 아는 것은 user 객체

Tell, Don't Ask

- 결국 로그인되었는지 아닌지 아는 것은 user 객체
  - 이 상태는 user 객체에 속한다.

Tell, Don't Ask

- 결국 로그인되었는지 아닌지 아는 것은 user 객체
  - 이 상태는 user 객체에 속한다.
  - 왜 user 상태를 가져다가,

Tell, Don't Ask

- 결국 로그인되었는지 아닌지 아는 것은 user 객체
  - 이 상태는 user 객체에 속한다.
  - 왜 user 상태를 가져다가,
  - user를 대신해서 결정하는가 ?

Tell, Don't Ask

- 결국 로그인되었는지 아닌지 아는 것은 user 객체
  - 이 상태는 user 객체에 속한다.
  - 왜 user 상태를 가져다가,
  - user를 대신해서 결정하는가 ?
  - user가 해당 행위를 수행하는 것이 맞다.

Tell, Don’t Ask

- 결국 로그인되었는지 아닌지 아는 것은 user 객체
  - 이 상태는 user 객체에 속한다.
  - 왜 user 상태를 가져다가,
  - user를 대신해서 결정하는가 ?
  - user가 해당 행위를 수행하는 것이 맞다.
- Tell Don't Ask

Tell, Don’t Ask

- 결국 로그인되었는지 아닌지 아는 것은 user 객체
  - 이 상태는 user 객체에 속한다.
  - 왜 user 상태를 가져다가,
  - user를 대신해서 결정하는가 ?
  - user가 해당 행위를 수행하는 것이 맞다.
- Tell Don't Ask
  - tell other object what to do,

Tell, Don’t Ask

- 결국 로그인되었는지 아닌지 아는 것은 user 객체
  - 이 상태는 user 객체에 속한다.
  - 왜 user 상태를 가져다가,
  - user를 대신해서 결정하는가 ?
  - user가 해당 행위를 수행하는 것이 맞다.
- Tell Don't Ask
  - tell other object what to do,
  - but not to ask object what the state is.

Tell, Don't Ask

- 결국 로그인되었는지 아닌지 아는 것은 user 객체
  - 이 상태는 user 객체에 속한다.
  - 왜 user 상태를 가져다가,
  - user를 대신해서 결정하는가 ?
  - user가 해당 행위를 수행하는 것이 맞다.
- Tell Don't Ask
  - tell other object what to do,
  - but not to ask object what the state is.
  - 객체의 상태를 얻어서 객체를 대신해서 결정을 하지 말고, 객체에게 결정하도록 하라.

Tell, Don't Ask

Tell, Don't Ask

```
o.getX()
    .getY()
        .getZ()
            .doSomething();
```

Tell, Don't Ask

```
o.getX()
    .getY()
        .getZ()
            .doSomething();
```

- Long chain of function

Tell, Don't Ask

```
o.getX()
    .getY()
        .getZ()
            .doSomething();
```

- Long chain of function
- Train wrecks

Tell, Don't Ask

```
o.getX()
    .getY()
        .getZ()
            .doSomething();
```

- Long chain of function
- Train wrecks
- Tell, Don't ask의 명확한 위반

Tell, Don't Ask

```
5 o.getX()
6     .getY()
7         .getZ()
8             .doSomething();
```

- Long chain of function
- Train wrecks
- Tell, Don't ask의 명확한 위반
  - 뭔가를 tell(doXXX)하기 전에 지속적으로 ask(getXXX)함.

Tell, Don't Ask

```
o.getX()
    .getY()
        .getZ()
            .doSomething();
```

- Long chain of function
- Train wrecks
- Tell, Don't ask의 명확한 위반
  - 뭔가를 tell(doXXX)하기 전에 지속적으로 ask(getXXX)함.

```
o.DoSomething();
```

Law of Demeter

- 하나의 함수가 전체 시스템의 객체들 간의 네비게이션을 아는 것은 나쁜 아이디어

Law of Demeter

- 하나의 함수가 전체 시스템의 객체들 간의 네비게이션을 아는 것은 나쁜 아이디어

```
o.getX()
    .getY()
        .getZ()
            .doSomething();
```

Law of Demeter

Law of Demeter

- 한 라인의 코드가 너무 방대한 지식을 알아야 함

Law of Demeter

- 한 라인의 코드가 너무 방대한 지식을 알아야 함
- 함수가 시스템에 너무 많이 의존성을 갖음

Law of Demeter

- 한 라인의 코드가 너무 방대한 지식을 알아야 함
- 함수가 시스템에 너무 많이 의존성을 갖음
- 함수가 우리 시스템의 전체를 알게 하고 싶지는 않다. 개별 함수는 아주 제한된 지식만 가져야 한다.

Law of Demeter

- 한 라인의 코드가 너무 방대한 지식을 알아야 함
- 함수가 시스템에 너무 많이 의존성을 갖음
- 함수가 우리 시스템의 전체를 알게 하고 싶지는 않다. 개별 함수는 아주 제한된 지식만 가져야 한다.
- 우리가 수행해야 할 것을 위해 이웃 객체에게 요청하길 원한다.

Law of Demeter

- 한 라인의 코드가 너무 방대한 지식을 알아야 함
- 함수가 시스템에 너무 많이 의존성을 갖음
- 함수가 우리 시스템의 전체를 알게 하고 싶지는 않다. 개별 함수는 아주 제한된 지식만 가져야 한다.
- 우리가 수행해야 할 것을 위해 이웃 객체에게 요청하길 원한다.
- 요청을 수신하면 적절한 객체가 수신할 때까지 전파되길 원한다.

Law of Demeter

Law of Demeter

- 규칙

Law of Demeter

- 규칙
  - You may call methods of objects that are

Law of Demeter

- 규칙
  - You may call methods of objects that are
    - Passed as arguments

Law of Demeter

- 규칙
  - You may call methods of objects that are
    - Passed as arguments
    - Created locally

Law of Demeter

- 규칙
  - You may call methods of objects that are
    - Passed as arguments
    - Created locally
    - instance variables

Law of Demeter

- 규칙
  - You may call methods of objects that are
    - Passed as arguments
    - Created locally
    - instance variables
    - globals

Law of Demeter

- 규칙
  - You may call methods of objects that are
    - Passed as arguments
    - Created locally
    - instance variables
    - globals
  - You may NOT call methods on objects that are:

Law of Demeter

- 규칙
  - You may call methods of objects that are
    - Passed as arguments
    - Created locally
    - instance variables
    - globals
  - You may NOT call methods on objects that are:
    - Returned from a previous method call.

Law of Demeter

- 규칙
  - You may call methods of objects that are
    - Passed as arguments
    - Created locally
    - instance variables
    - globals
  - You may NOT call methods on objects that are:
    - Returned from a previous method call.
    - o.getX().getY().getZ().doSomething()

Law of Demeter

- 규칙
  - You may call methods of objects that are
    - Passed as arguments
    - Created locally
    - instance variables
    - globals
  - You may NOT call methods on objects that are:
    - Returned from a previous method call.
    - o.getX().getY().getZ().doSomething()
- Ask대신 Tell하면 surrounding과 decouple된다.

Law of Demeter

- 규칙
  - You may call methods of objects that are
    - Passed as arguments
    - Created locally
    - instance variables
    - globals
  - You may NOT call methods on objects that are:
    - Returned from a previous method call.
    - o.getX().getY().getZ().doSomething()
- Ask대신 Tell하면 surrounding과 decouple된다.
- 세포들은 다른 세포들에게 뭔가를 묻지 않는다.

Law of Demeter

- 규칙
  - You may call methods of objects that are
    - Passed as arguments
    - Created locally
    - instance variables
    - globals
  - You may NOT call methods on objects that are:
    - Returned from a previous method call.
    - o.getX().getY().getZ().doSomething()
- Ask대신 Tell하면 surrounding과 decouple된다.
- 세포들은 다른 세포들에게 뭔가를 묻지 않는다.
- 대신 원하는 것을 하도록 말한다.