6. Design: Design Principles

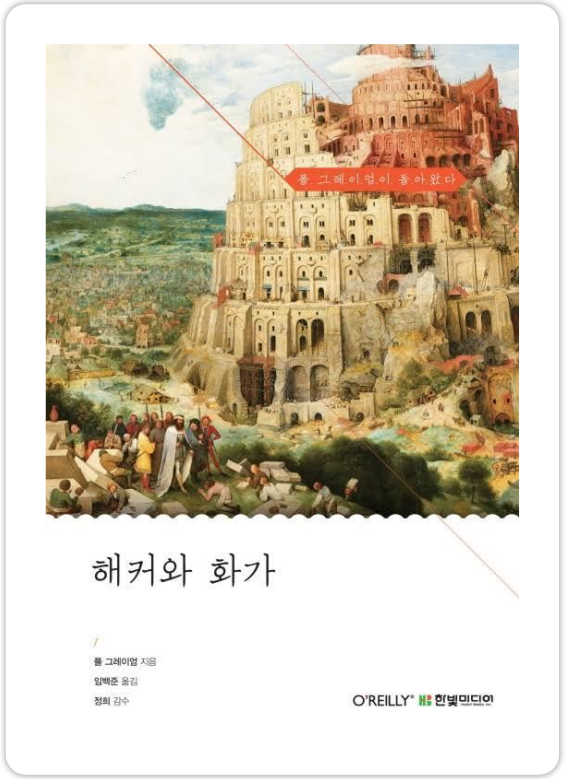
Choi, Kwanghoon

Chonnam National University

Table of Contents

6.4 Design Principles

- What is a good design?
  - (참고) 해커와 화가 9장
- Step-wise refinement
- Abstraction
- Modularization
  - Coupling, Cohesion
- SOLID: OO Design Principles



Step-wise Refinement: Example

- A Step-wise Refinement Design of Tic-tac-toe

단계	Level 1
[1]	게임을 초기화하기
[2]	승부가 결정되거나 무승부로 종료될 때까지 번갈아 말을 놓기
[3]	최종 게임 결과를 출력하기

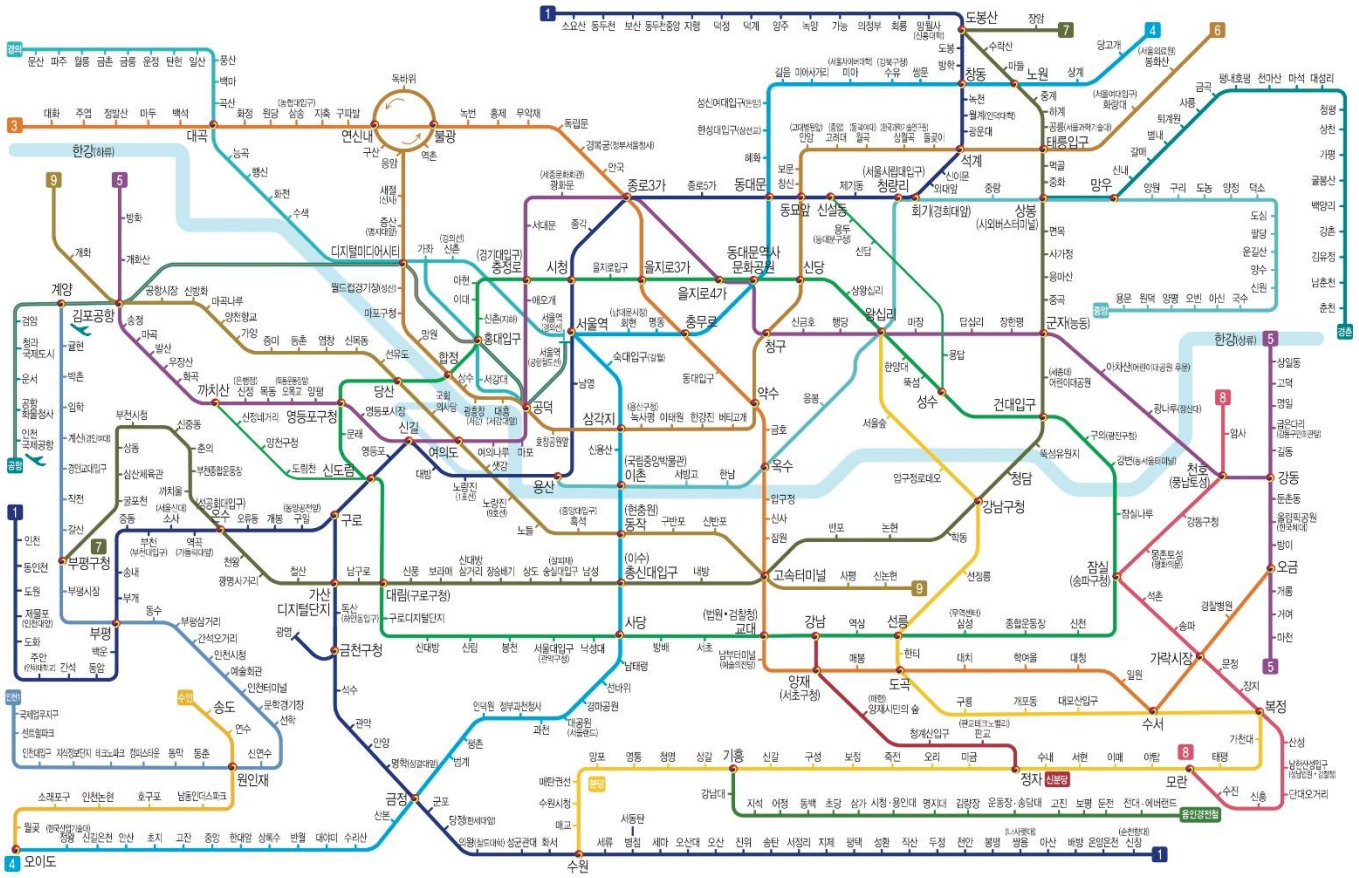
Step-wise Refinement: Example

- A Step-wise Refinement Design of Tic-tac-toe (cont.)

단계	Level 1	Level 2
[1]	게임을 초기화하기	1-1) 보드를 초기화하기 1-2) 현재 차례 초기화하기
[2]	승부가 결정되거나 무승부로 종료될 때까지 번갈아 말을 놓기	2-1) 현재 보드를 출력하기 2-2) 이번 차례의 말을 입력받아 놓기 2-3) 아래와 같이 승부 판단하기 2-3-1) 이번 차례가 승리했다면 [3]으로 이동 2-3-2) 아직 승부가 결정되지 않았다면 다음을 실행 2-3-2-1) 더 둘 곳이 남아 있으면 다음 차례로 바꾸고 2-1)로 이동 2-3-2-2) 더 둘 곳이 없으면 [3]으로 이동
[3]	최종 게임 결과를 출력하기	3-1) 승부가 결정되었다면 이번 차례가 이겼음 3-2) 승부가 결정되지 않았다면 무승부임

Abstraction(추상화)

- Show important things (Hide unimportant things)



Abstraction(추상화)

- Procedural abstractions
  - Show procedure interfaces
  - Hide internal algorithms

- Data abstractions
  - Show interfaces/operations on data
  - Hide data representations(e.g., array, list, map, …)

  - cf. Information hiding, Encapsulation, Abstract data types

Modularization

[교재 참고] 결합도에 영향을 주는 요소

- Coupling(두 모듈 사이의 결합도)
  - A measure of how closely connected two routines are
  - Data, Stamp, Control, Common, Content coupling

Good/Weak ⟷ *Bad/Strong*

- Cohesion(하나의 모듈 내에서 응집도)
  - The strength of relationship between pieces of functionality with a given module
  - Function, Sequential, Communication, Temporal, Logical, Coincidental cohesion

Good/Strong ⟷ *Bad/Weak*

Modularization: Coupling

자료 결합 (data coupling)	모듈 간의 인터페이스가 매개 변수 전달 등의 자료 요소로만 구성된 경우
스탬프 결합 (stamp coupling)	배열이나 레코드를 전달하되 그 중 일부만 전달하는 경우. 전역 변수 일부만을 사용하여 전달하는 경우
제어 결합 (control coupling)	제어 흐름 관련 값을 전달하여 다른 모듈의 실행 순서를 제어하는 경우
공통 결합 (common coupling)	여러 모듈이 공동 자료 영역을 사용하는 경우
내용 결합 (content coupling)	다른 모듈의 지역 변수나 명령어를 수정하는 경우

Modularization: Cohesion

기능적 응집 (functional cohesion)	모듈이 하나의 기능을 구현하는 내용으로만 작성됨
순차적 응집 (sequential cohesion)	모듈 안에서 하나의 소작업 결과가 다음 소작업의 입력으로 전달되는 형태
커뮤니케이션 응집 (communication cohesion)	동일한 입출력 데이터에 대한 오퍼레이션을 모아놓은 형태 (수행 순서와는 무관)
시간적 응집 (temporal cohesion)	동일한 시점에 수행되는 오퍼레이션을 모아놓은 형태 (예: 초기화 코드를 모두 모아놓은 모듈)
논리적 응집 (logical cohesion)	모듈의 요소들이 논리적으로 관련있어서 모아놓은 형태 (예: 수학함수 라이브러리)
이유없는 응집 (coincidental cohesion)	관계가 없는 코드들이 하나의 모듈에 있는 형태

SOLID

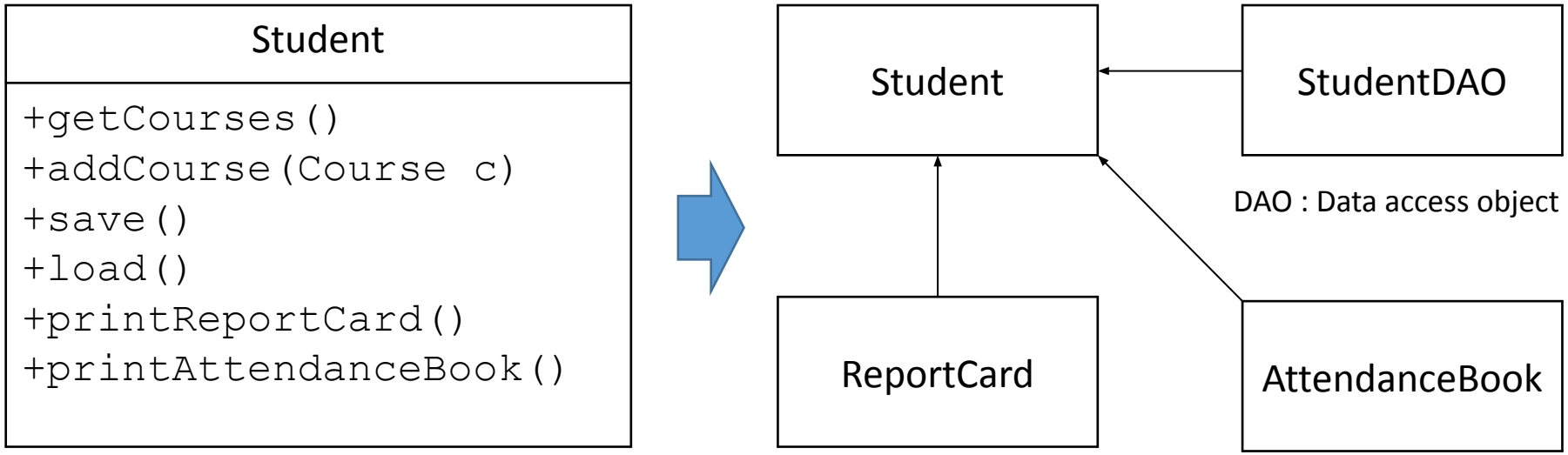
- Single Responsibility Principle
  - (SRP, 단일 책임 원리)
- Open-Closed Principle
  - (OCP, 개방-폐쇄 원리)
- Liskov Substitution Principle
  - (LSP, 리스코프 치환 원리)
- Interface Segregation Principle
  - (ISP, 인터페이스 분리 원리)
- Dependency Inversion Principle
  - (DIP, 의존 역전 원리)

참고) 정인상, 채흥석, Java 객체지향 디자인 패턴, 한빛미디어

SOLID: SRP (단일 책임 원리)

- Single Responsibility Principle (SRP)
  - Every module (or class) should have responsibility over a single part of the functionality.

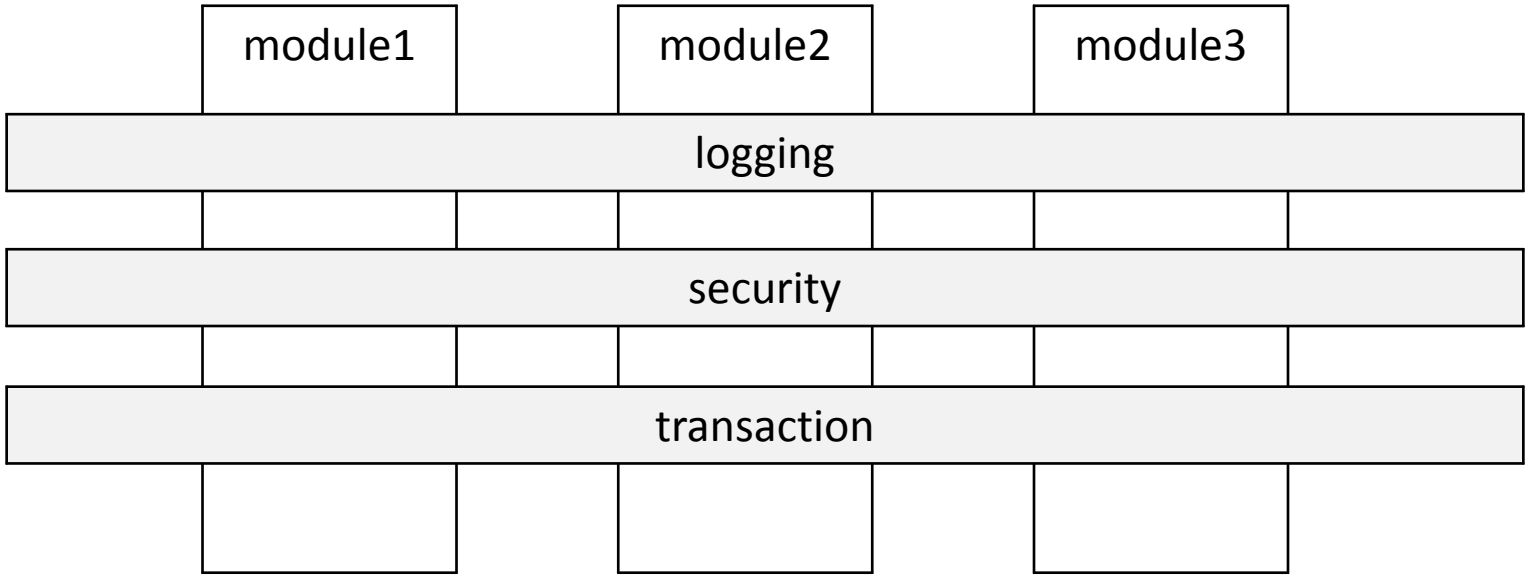
[Case1: a module with multiple responsibilities]



SOLID: SRP (단일 책임 원리)

- Single Responsibility Principle (SRP)
  - Every module (or class) should have responsibility over a single part of the functionality.

[Case2: multiple modules sharing a single responsibility]
(e.g., logging, security, transactions)

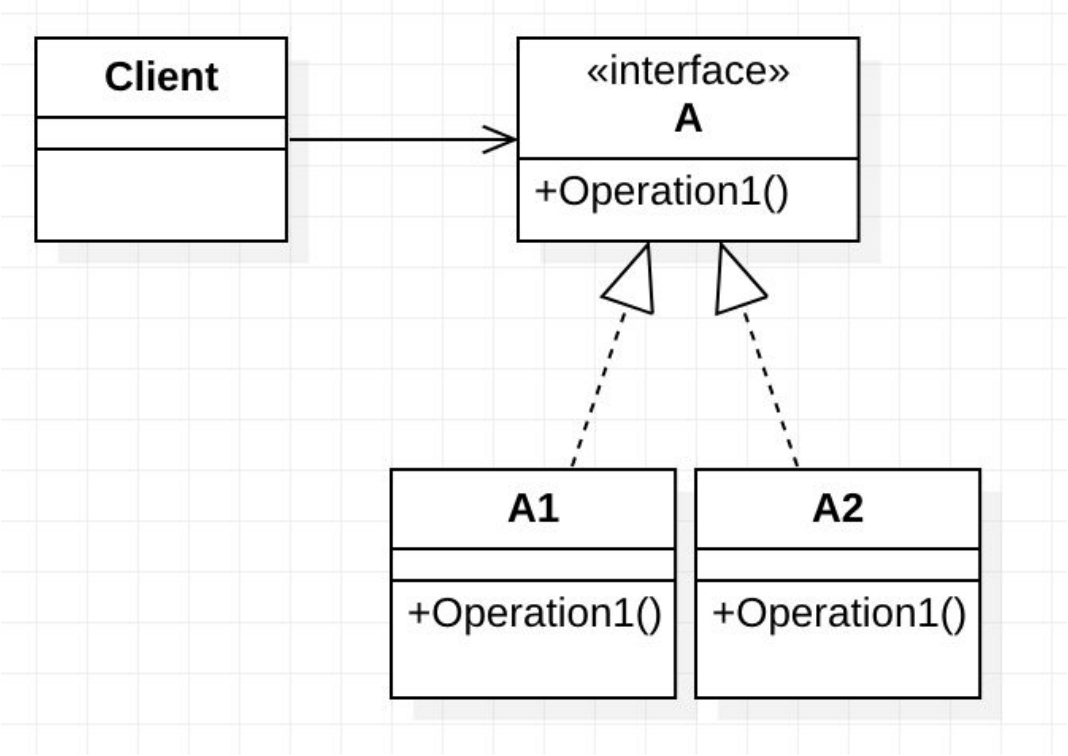


cf. aspect-oriented programming, AspectJ

SOLID: OCP (개방 폐쇄 원리)

- Open-Closed Principle (OCP)
  - Every module can allow its behavior to be extended without modifying its source code.

(새로운 확장은 허용하되(Open), 기존 코드는 수정하지 않도록(Closed) 설계)



SOLID: OCP (개방 폐쇄 원리)

- Open-Closed Principle (OCP)
  - Every module can allow its behavior to be extended without modifying its source code.

(새로운 확장은 허용하되(Open), 기존 코드는 수정하지 않도록(Closed) 설계)

Q. Redesign the following code to make suitable both for deployment and testing using OCP.

```
import java.util.Calendar;
public class TimeReminder {
  private MP3 m;
  public void reminder() {
   Calendar cal=Calendar.getInstance();
   m = new MP3();
   int hour = cal.get(Calendar.HOUR_OF_DAY);
   if(hour >= 22) {
      m.play();
   }
  }
}
```

SOLID: LSP (리스코프 치환 원리)

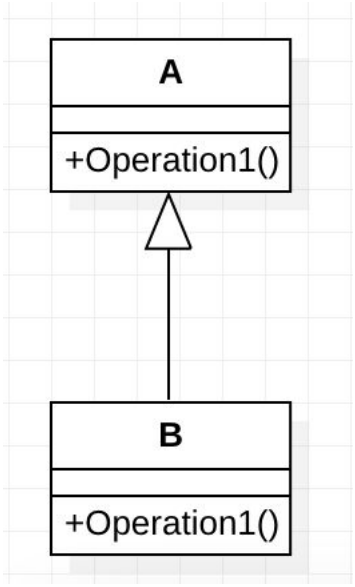
- Liskov Substitution Principle
  - Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program

Babara Liskov, MIT
(Turing award 2008 : Design of OO PLs)

SOLID: LSP (리스코프 치환 원리)

- Liskov Substitution Principle
  - Behavioral subtyping
    - Subclasses should satisfy the *behavioral expectations* of clients (*specifications of super classes)* accessing subclass objects through references of superclass type

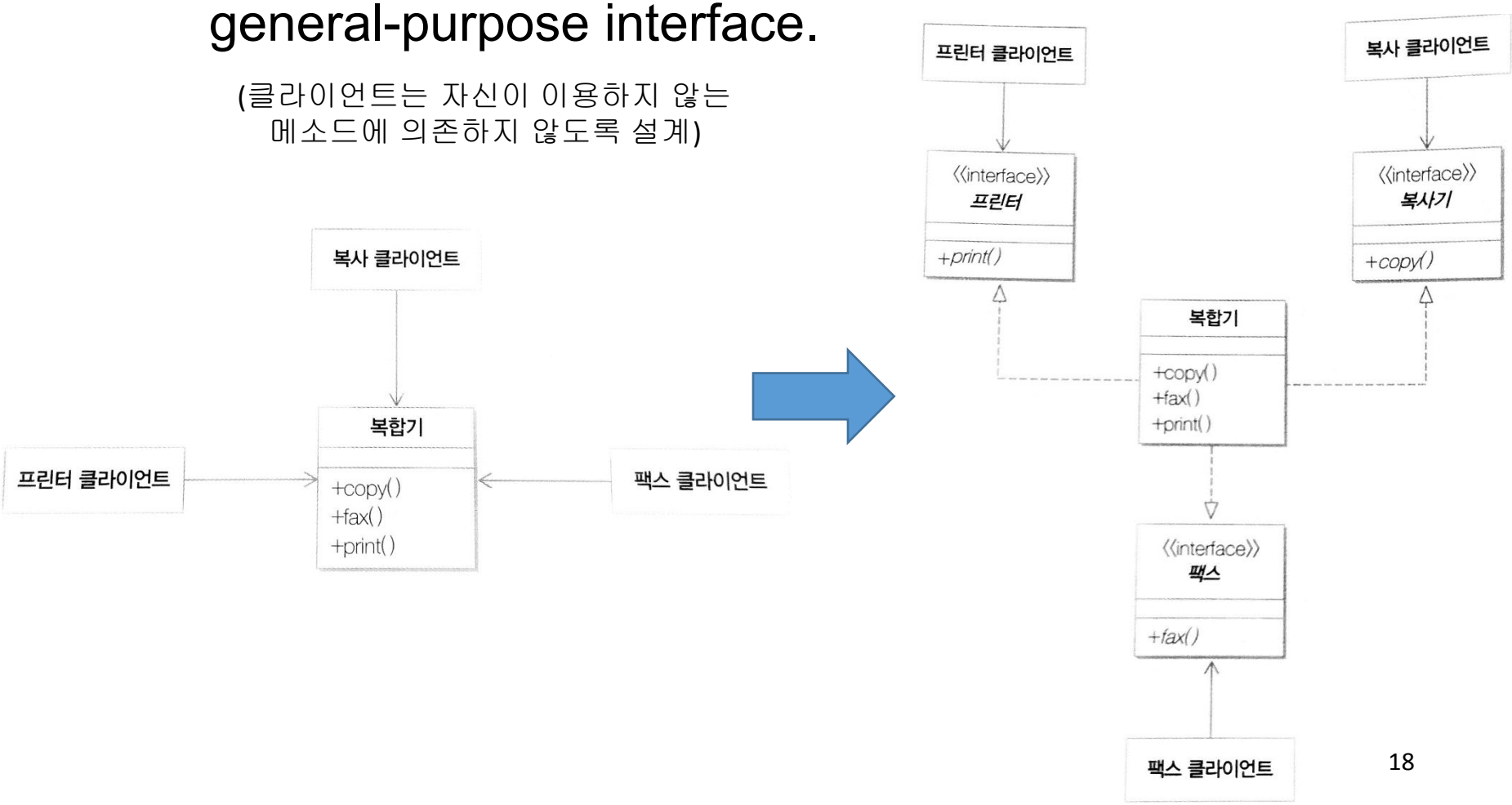
```
public class client {
    …
    void client_method(A a) {
        …
        a.operation1();
    }
}

Client c = new Client();
c.client_method(new B());
```



SOLID: ISP (인터페이스 분리 원리)

- Interface Segregation Principle
  - Many client-specific interfaces are better than one general-purpose interface.

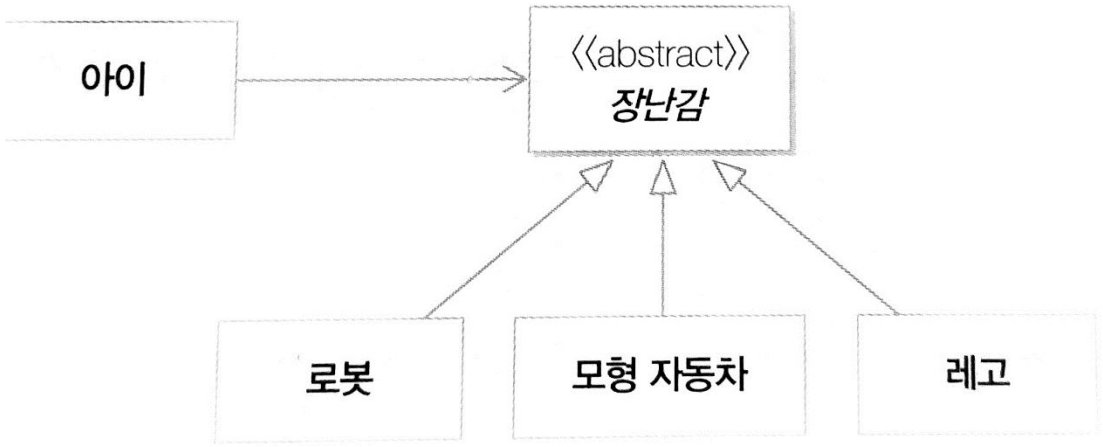
(클라이언트는 자신이 이용하지 않는 메소드에 의존하지 않도록 설계)

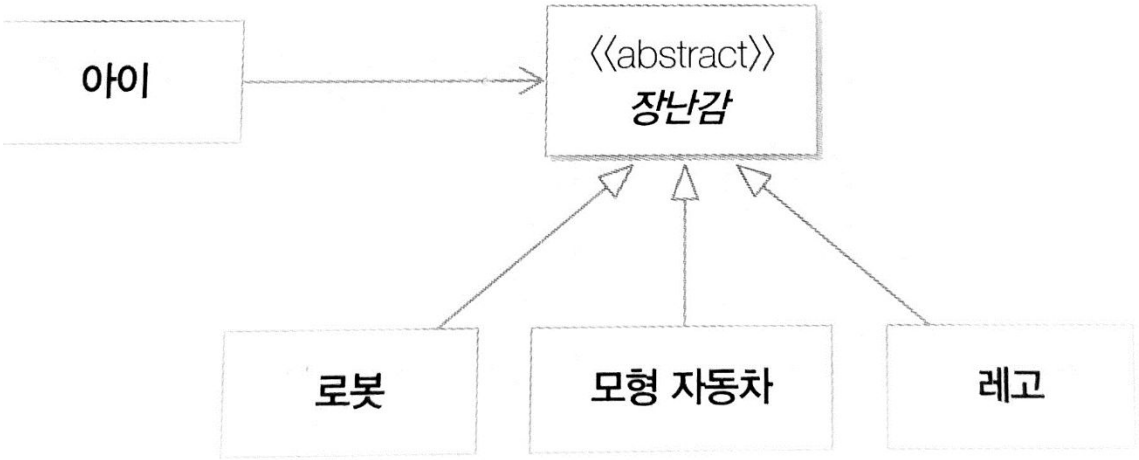


SOLID: DIP (의존 역전 원리)

- Dependency Inversion Principle
  - One should depend on abstractions, not concretions.

    (변하지 않는 것에 의존하도록 설계하는 원리)






```
class Kid {
 private Toy toy;
 void setToy(Toy toy) { this.toy = toy; }
 void play() { System.out.println(toy.toString()); }
}

Kid k = new Kid();
k.setToy(new Robot());
k.play();

k.setToy(new Lego());
k.play();
```

의존성 주입(dependency injection):
- 예) Kid 객체가 사용할 Toy 객체를 외부에서 변경