6주 1강

소프트웨어 하위 설계

이번 주차에는…

소프트웨어 하위 설계

- 모듈 설계
- 소프트웨어 개발 방법과 설계
- 객체지향의 주요 개념과 특징
- 클래스간의 관계와 설계 원칙

1. 모듈과 모듈화의 이해(1)

- 요구 분석 단계
  - DFD(구조적 방법), ERD(정보공학 방법), usecase diagram(객체지향 방법)
    → 요구 분석 명세서

- 상위 설계(아키텍처 설계)
  - 전체 구조를 파악하여 표현

- 하위 설계(아키텍처 설계)
  - 상세한 내용을 다루는 모듈 설계

2. 모듈과 모듈화의 이해(2)

- 모듈화
  - 소프트웨어 개발에서 큰 문제를 작은 단위로 나누는 것

- 모듈
  - '규모가 큰 것을 여러 개로 나눈 조각'
  - '소프트웨어 구조를 이루는 기본적인 단위'
  - '하나 또는 몇 개의 논리적인 기능을 수행하기 위한 명령어들의 집합'
  - 독립 프로그램도 하나의 모듈, 함수(메서드)들도 하나의 모듈

3. 모듈과 모듈화의 이해(3)

- 모듈화의 특징
  - 다른 것들과 구별될 수 있는 독립적인 기능을 갖는 단위이다.
  - 유일한 이름이 있어야 한다.
  - 독립적으로 컴파일이 가능하다.
  - 모듈에서 또 다른 모듈을 호출할 수 있다.
  - 다른 프로그램에서도 모듈을 호출할 수 있다.

- 모듈화의 형태
  - 용도가 비슷한 것끼리 묶어놓은 라이브러리 함수, 그래픽 함수
  - 추상화된 자료, subroutine, procedure, object, method

4. 모듈과 모듈화의 이해(4)

- 좋은 모듈 설계를 위한 원칙
  - 모듈 간의 결합coupling은 느슨하게loosely
  - 모듈 내 구성 요소들 간의 응집cohesion은 강하게strongly
- 모듈화의 장점
  - 분할과 정복divide and conquer의 원리가 적용되어 복잡도 감소한다.
  - 문제를 이해하기 쉽게 만든다.
  - 변경하기 쉽고, 변경으로 인한 영향이 적다.
  - 유지보수가 용이하다.
  - 프로그램을 효율적으로 관리할 수 있다.
  - 오류로 인한 파급효과를 최소화할 수 있다.
  - 설계 및 코드를 재사용할 수 있다.

5. 모듈화와 소프트웨어 개발 비용

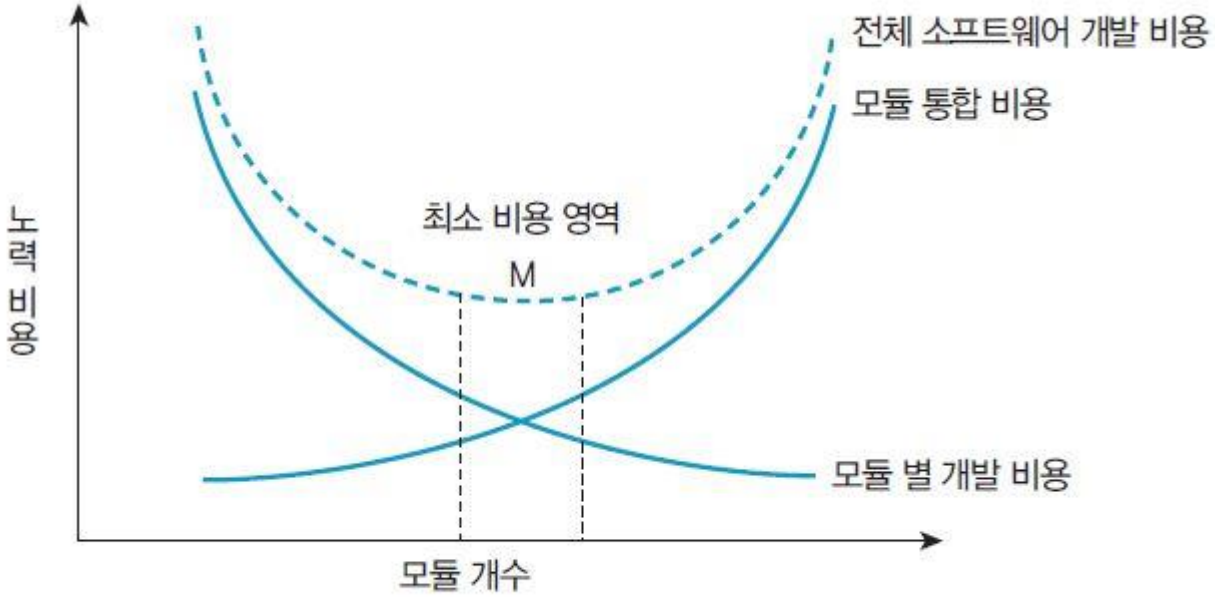

그림 6-1 모듈화와 소프트웨어 개발 비용

6. 응집도

- 응집도cohesion
  - 모듈 내부에 존재하는 구성 요소들 사이의 밀접한 정도
  - 하나의 모듈 안에서 구성 요소들 간에 똘똘 뭉쳐 있는 정도로 평가

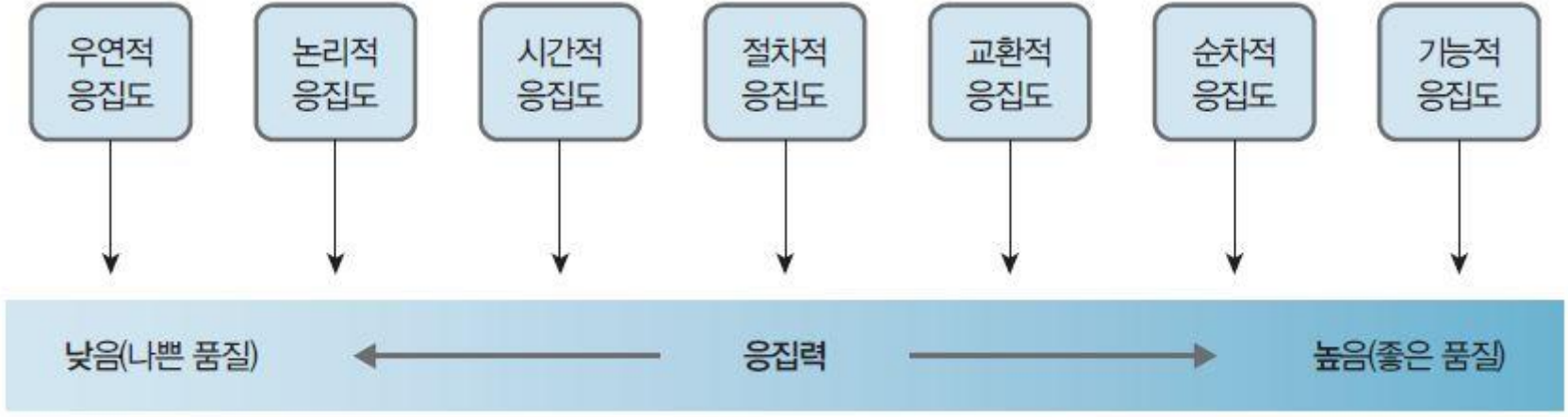

그림 6-2 모듈 내 구성 요소 간의 응집도

7. 기능적 응집

- 기능적 응집functional cohesion
  - 함수적 응집
  - 응집도가 가장 높은 경우이며 단일 기능의 요소로 하나의 모듈을 구성

그림 6-3 기능적 응집의 예

8. 순차적 응집

- 순차적 응집sequential cohesion
  - [그림 6-4]처럼 A 요소의 출력을 B 요소의 입력으로 사용하므로 두 요소가 하나의 모듈을 구성한 경우
  - 두 요소가 아주 밀접하므로 하나의 모듈로 묶을 만한 충분한 이유가 된다.

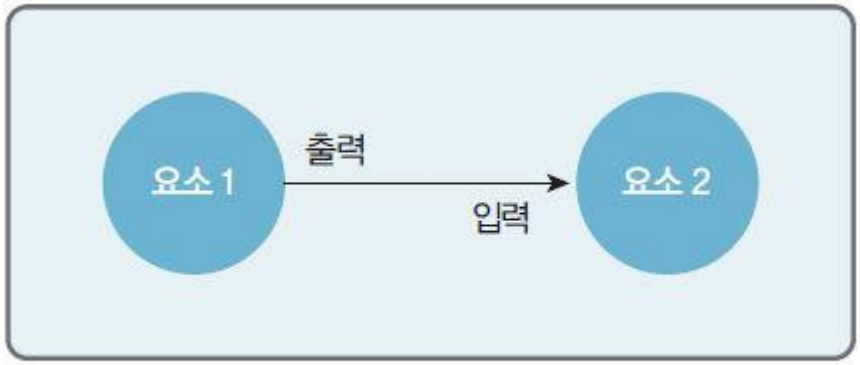

그림 6-4 순차적 응집의 예

9. 교환적 응집

- 교환적 응집communication cohesion
  - 정보적 응집
  - [그림 6-5]처럼 같은 입력을 사용하는 구성 요소들을 하나의 모듈로 구성
  - 구성 요소들이 동일한 출력을 만들어낼 때도 교환적 응집
  - 요소들 간의 순서는 중요하지 않다.

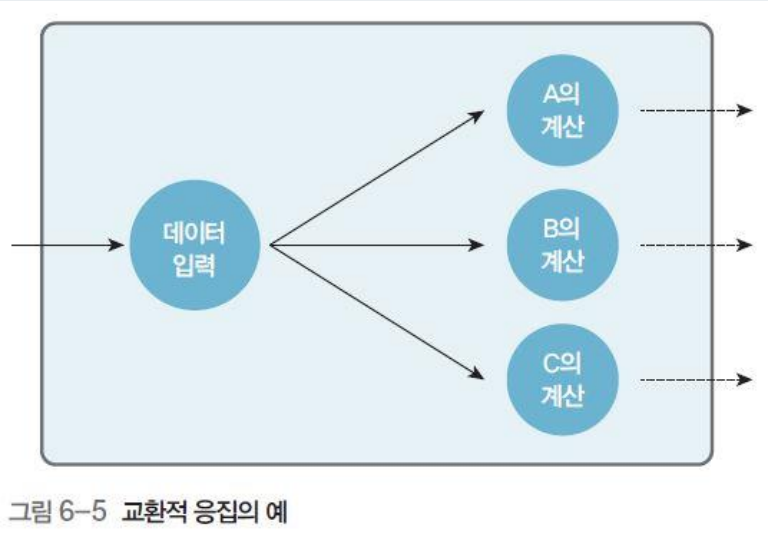

그림 6-5 교환적 응집의 예

10. 절차적 응집

- 절차적 응집procedural cohesion
  - [그림 6-6]처럼 순서가 정해진 몇 개의 구성 요소를 하나의 모듈로 구성
  - 순차적 응집과 다른 점: 어떤 구성 요소의 출력이 다음 구성 요소의 입력으로 사용되지 않고, 순서에 따라 수행만 된다는 것

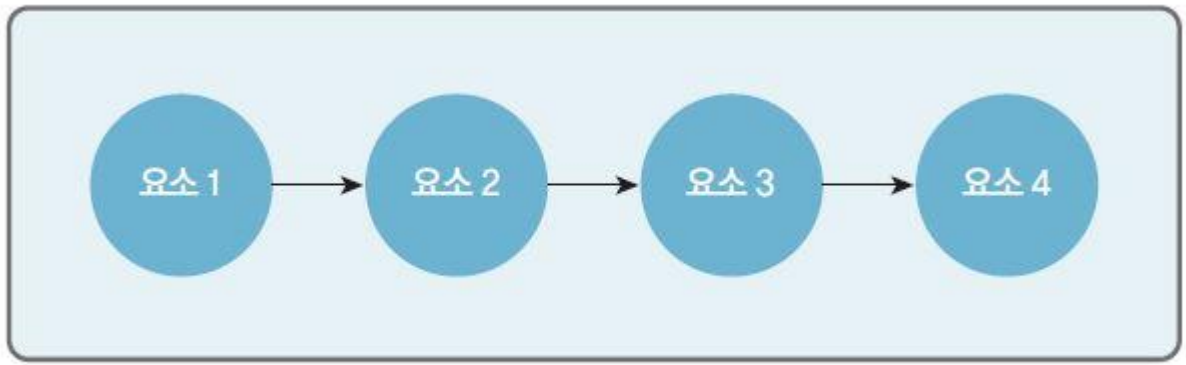

그림 6-6 절차적 응집의 예

11. 시간적 응집

- 시간적 응집temporal cohesion
  - 모듈 내 구성 요소들의 기능도 다르고, 한 요소의 출력을 입력으로 사용하는 것도 아니고, 요소들 간에 순서도 정해져 있지 않다.
    But!
  - 그 구성 요소들이 같은 시간대에 함께 실행된다는 이유로 하나의 모듈로 구성

그림 6-7 시간적 응집의 예

12. 논리적 응집

- 논리적 응집logical cohesion
  - 모듈 간 순서와 무관, 한 모듈의 출력을 다른 모듈의 입력으로 사용하는 것도 아님
    But!
  - 요소들 간에 공통점이 있거나 관련된 임무가 존재하거나 기능이 비슷하다는 이유로 하나의 모듈로 구성

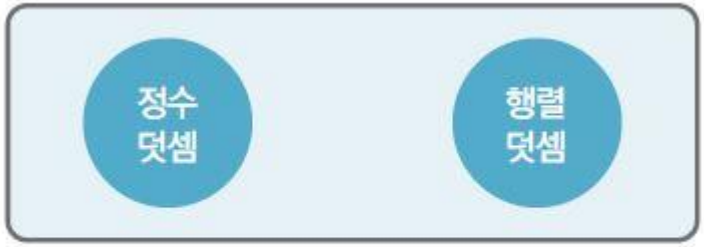

그림 6-8 논리적 응집의 예

- 비슷한 기능(입출력) : scanf( ), printf( )를 결합시킨 입출력 모듈
- 공통점(덧셈) : 정수의 덧셈과 행렬의 덧셈을 결합시킨 덧셈 모듈
- 공통점(출력) : 단말기 출력 기능과 파일 출력 기능을 결합시킨 출력 모듈

13. 우연적 응집

- 우연적 응집coincidental cohesion
  - 구성 요소들이 말 그대로 우연히 모여 구성
  - 특별한 이유 없이, 크기가 커 몇 개의 모듈로 나누는 과정에서 우연히 같이 묶인 것

그림 6-7 시간적 응집의 예

14. 결합도(1)

- 좋은 관계와 나쁜 관계

그림 6-10 결합도에 따른 좋은 관계와 나쁜 관계의 예

15. 결합도(2)

- 결합도coupling
  - 모듈과 모듈 사이의 관계에서 관련 정도
  - 하나의 모듈 안에서 구성 요소들 간에 똘똘 뭉쳐 있는 정도로 평가
  - 좋은 설계: loosely coupled
    ∵ 상호 의존성이 줄어 모듈의 독립성이 높아지고, 모듈 간에 영향이 적기 때문

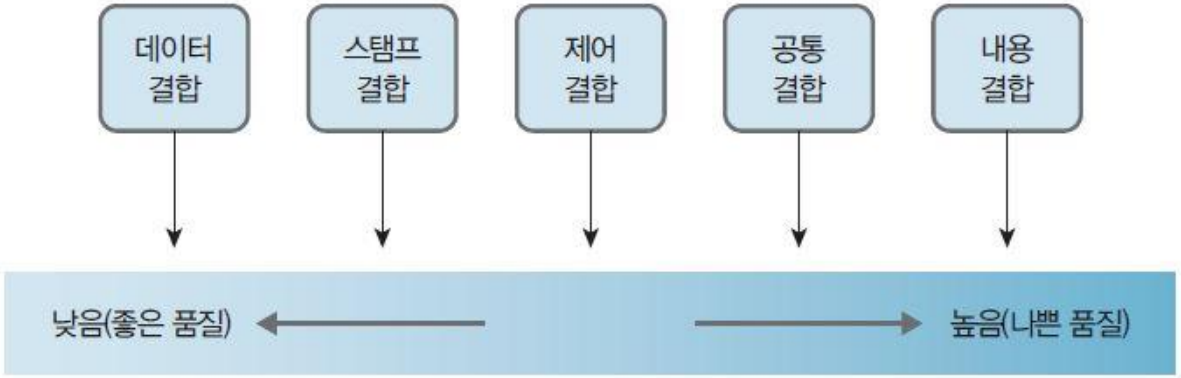

그림 6-11 모듈 간의 결합도

16. 데이터 결합

- 데이터 결합data coupling
  - 가장 좋은 모듈 간 결합
  - 모듈들이 매개변수를 통해 데이터만 주고받음으로써 서로 간섭을 최소화하는 관계
  - 모듈 간의 독립성 보장
  - 관계가 단순해 하나의 모듈을 변경했을 때 다른 모듈에 미치는 영향이 아주 적음

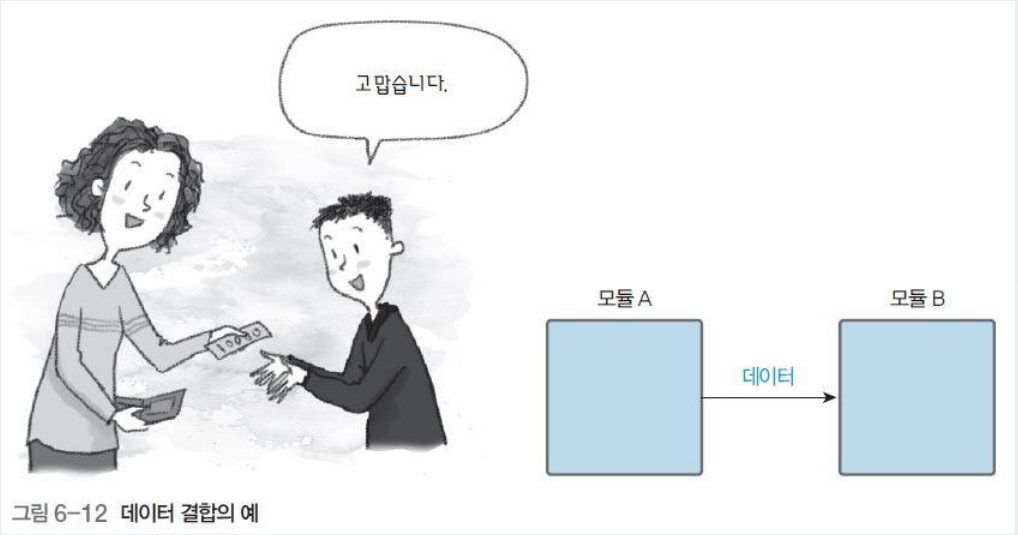

그림 6-12 데이터 결합의 예

17. 스탬프 결합

- 스탬프 결합stamp coupling

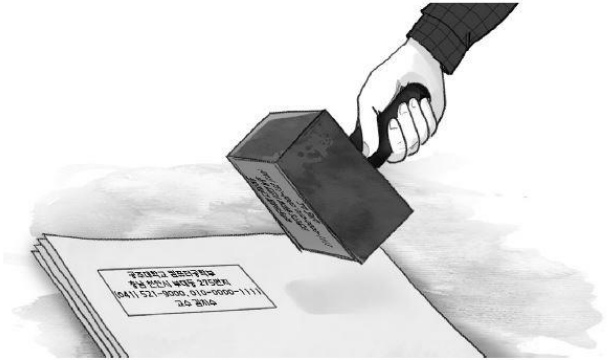

그림 6-13 스탬프의 예

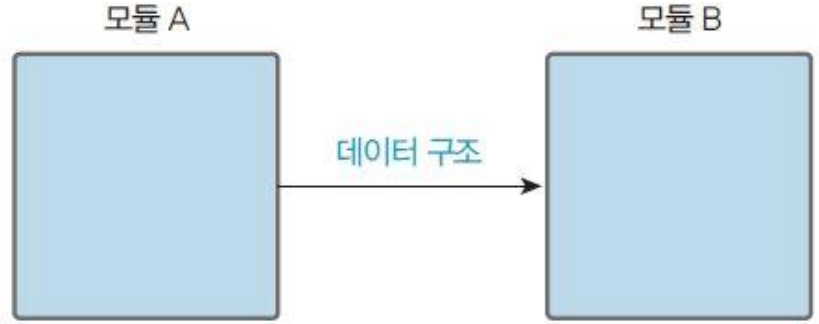

그림 6-14 스탬프 결합의 예

- 두 모듈 사이에서 정보를 교환할 때 필요한 데이터만 주고받을 수 없고 스탬프처럼 필요 없는 데이터까지 전체를 주고받아야 하는 경우
- 레코드나 배열 같은 데이터 구조, C 언어의 구조체(struct)

18. 제어결합

- 제어 결합control coupling

그림 6-15 제어 결합의 일상 예

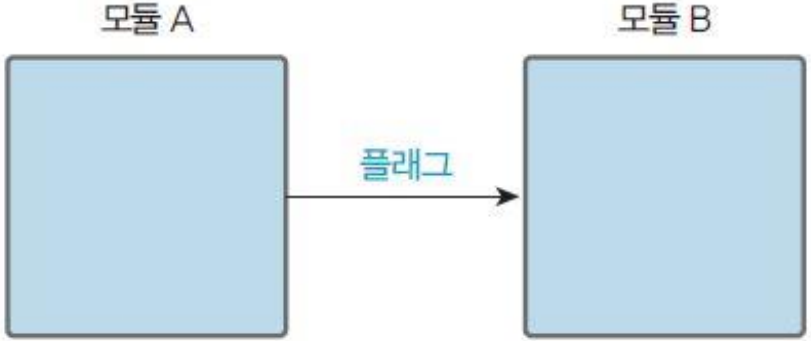

그림 6-16 제어 결합의 예

- 제어 플래그flag를 매개변수로 사용하여 간섭하는 관계
- 호출하는 모듈이 호출되는 모듈의 내부 구조를 잘 알고 논리적 흐름을 변경하는 관계
- 정보은닉을 크게 위배하는 결합으로, 다른 모듈의 내부에 관여하여 관계가 복잡해진다.

19. 공통 결합

- 공통 결합common coupling

그림 6-17 공통 결합의 일상 예

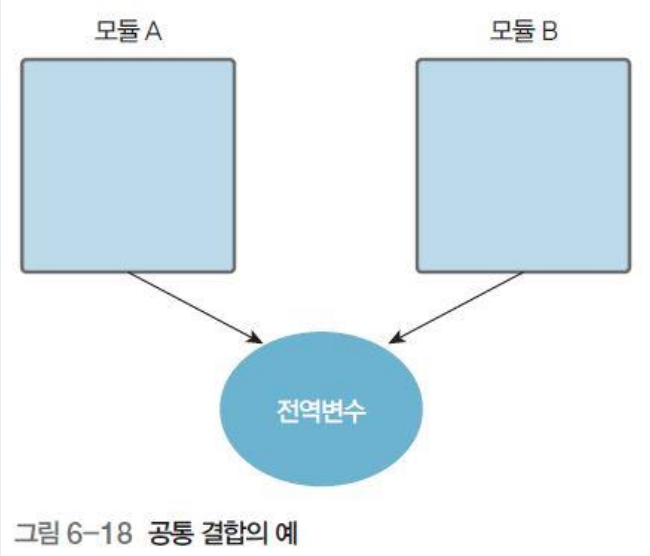

그림 6-18 공통 결합의 예

- 모듈들이 공통 변수(전역변수)를 같이 사용하여 발생하는 관계
- 문제점: 변수 값이 변하면 모든 모듈이 함께 영향을 받는다는 것

20. 내용 결합

- 내용 결합content coupling

그림 6-19 내용 결합의 개념 예

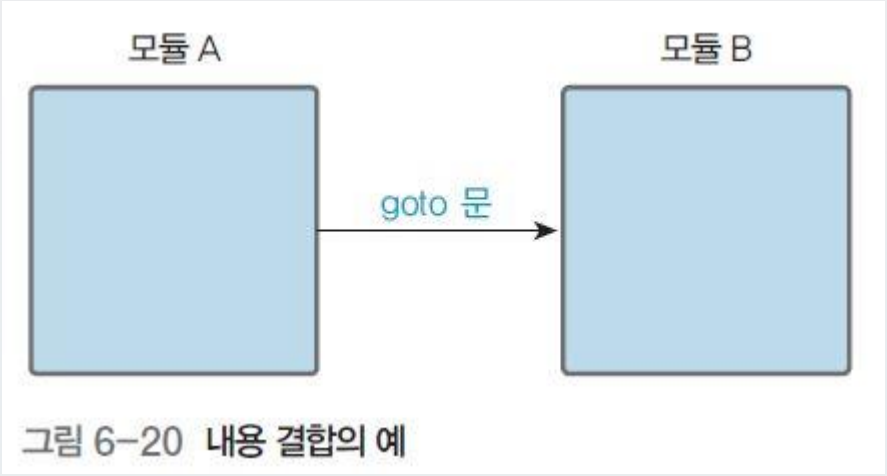

그림 6-20 내용 결합의 예

- 모듈 간에 인터페이스를 사용하지 않고 직접 왔다 갔다 하는 경우의 관계
- 상대 모듈의 데이터를 직접 변경할 수 있어 서로 간섭을 가장 많이 하는 관계
- C 언어의 goto 문

21. 모듈 간의 좋은 관계

- 바람직한 설계
  - 모듈 간에는 꼭 필요한 데이터만 주고받도록
  - 적은 인터페이스의 수를 통한 약한 결합 유지
  - 매개변수로 제어 플래그보다 데이터를 사용 → 유지보수 용이성 향상

낮은 결합도!

높은 응집도!

다음 시간

객체지향의 개발 방법