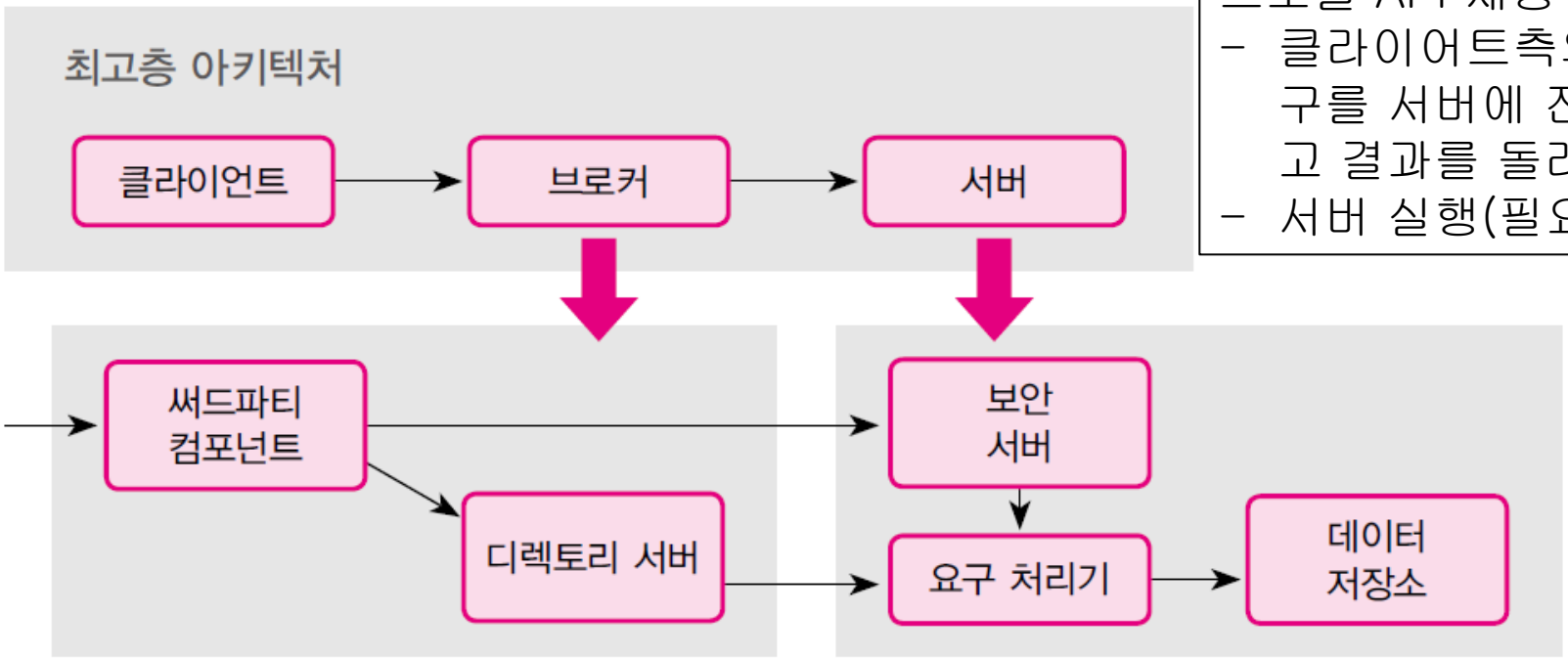
아키텍처의 표현

- **아키텍처의 계층적 분할**

Broker:
-서버등록 및 서버서비스호출 API 제공
- 클라이어트측의 요구를 서버에 전달 하고 결과를 돌려줌
- 서버 실행(필요시)



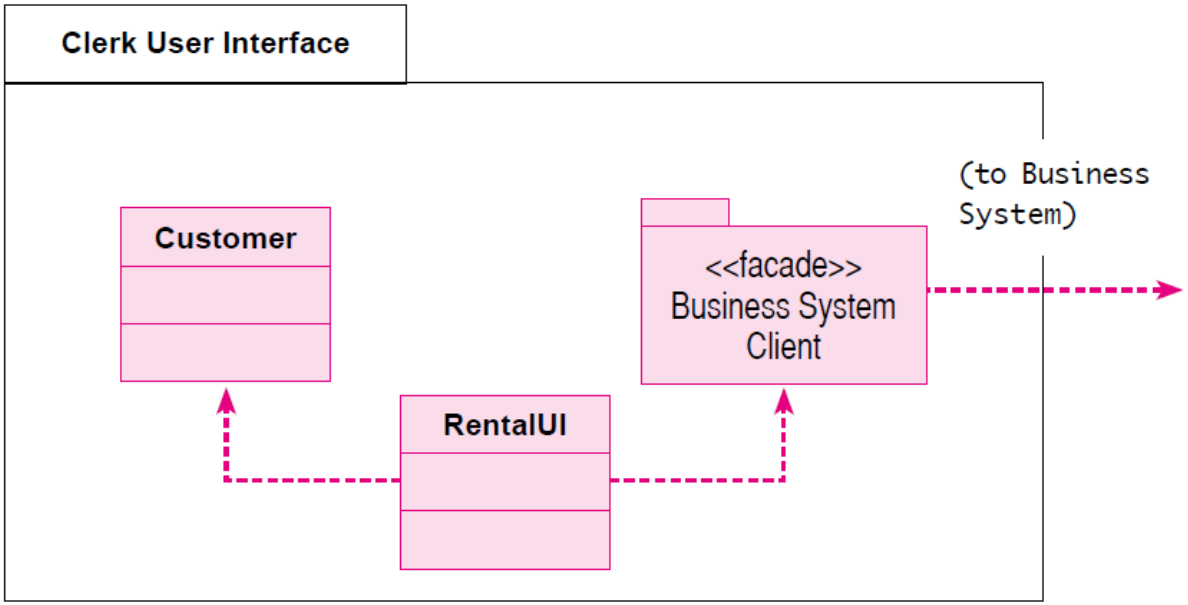
패키지 다이어그램

- UML에서 서브시스템을 표현할 때 도입되는 개념
  - 패키지 A modeling element to group modeling elements, and to provide a namespace for the grouped elements
  - 클래스를 의미 있는 관련된 그룹으로 구성하는 메커니즘
  - Java의 Package 선언으로 매핑
  - 패키지는 중첩될 수 있음
  - 복잡한 시스템을 서브시스템으로 나누어 적절한 컨트롤 가능
  - 소프트웨어 구조를 표현하는데 적합
  - 서브시스템으로 분할하면 객체 사이의 의존성을 최소화할 수 있어 솔루션 도메인의 복잡성을 줄일 수 있음
  - Façade 패턴

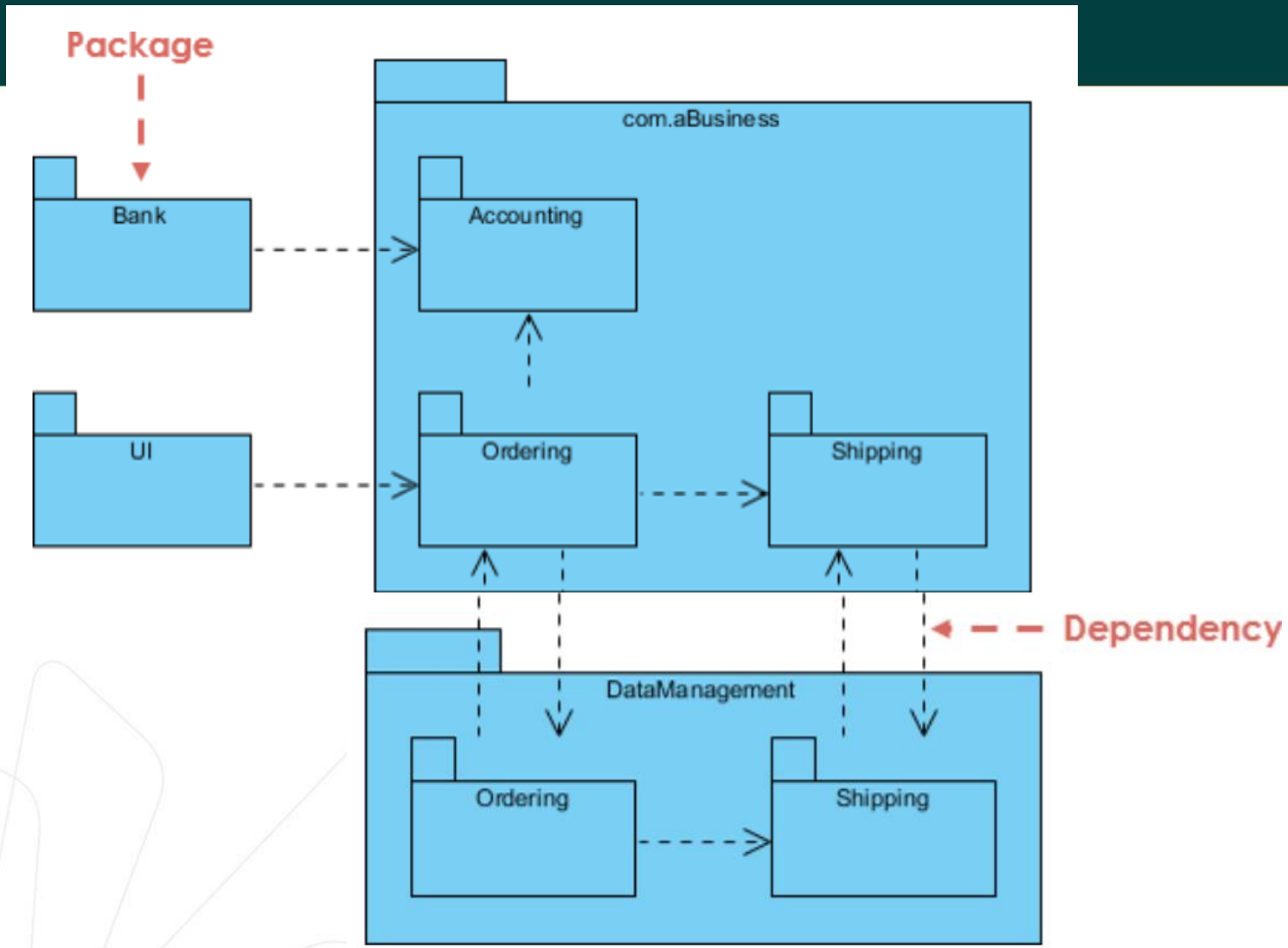
패키지 다이어그램

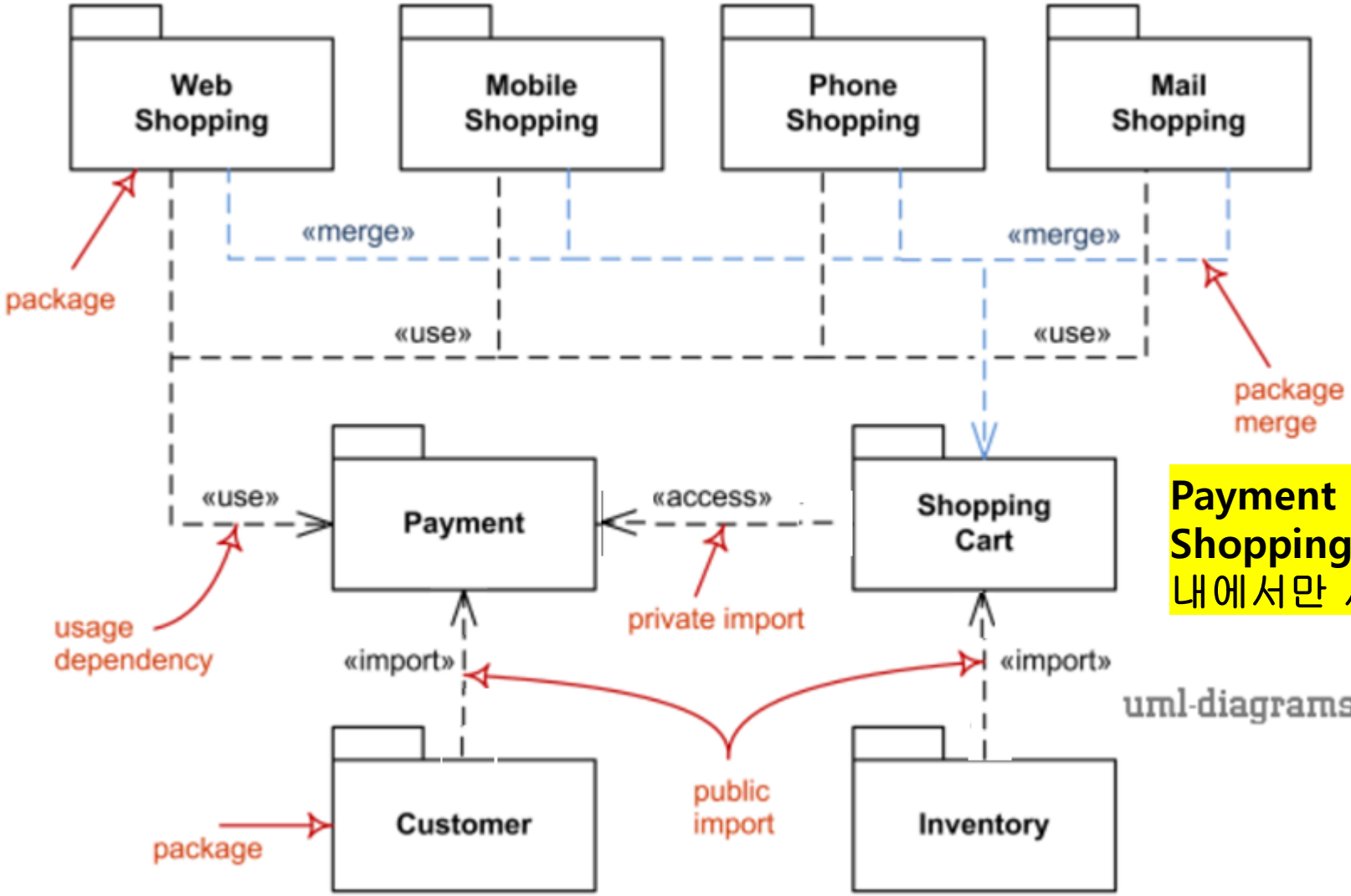
- 패키지 다이어그램에서의 의존 관계

- 패키지로 정의한 서브 시스템



- 객체를 서브시스템으로 grouping하는 heuristics
  - 같은 사용 사례에서 발견된 객체들은 같은 서브시스템에
  - 서브시스템들간의 데이터전달을 담당하는 객체는 별도 서브시스템으로
  - 서브시스템들간의 연관관계를 최소화
  - 같은 서브시스템내 객체들은 가능적으로 연관되어 있어야 함.


Package
com.aBusiness
Bank
Accounting
UI
Ordering
Shipping
Dependency
DataManagement
Ordering
Shipping

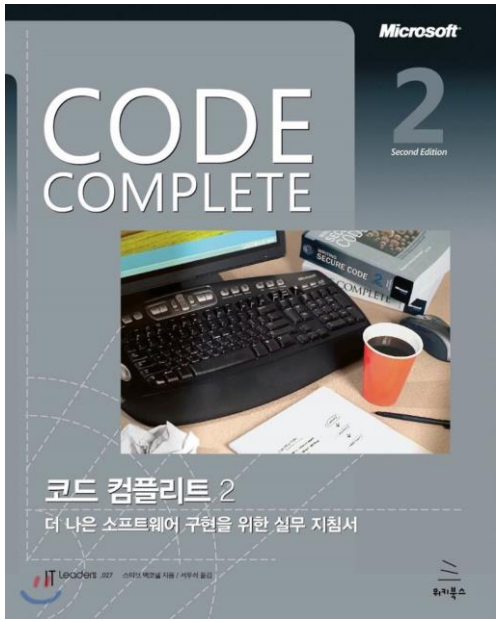


Payment 패키지는 **Shopping cart** 패키지 내에서만 사용 가능

Payment 패키지는 **Customer cart** 패키지 를 **import** 하는 타패키지에서 사용 가능

6.2 설계 원리: 좋은 설계 (from Code Complete 2)

- 7대 설계 목표
  - 복잡성 최소화 ➔ 분할, 추상화
  - 느슨한 결합 (loose coupling)
  - 강한 응집 (strong cohesion)
  - 확장성
  - 재사용성
  - 유지보수성
  - 유연성 ➔ 하드웨어/플랫홈/운영체게 변경시 쉽게 포팅



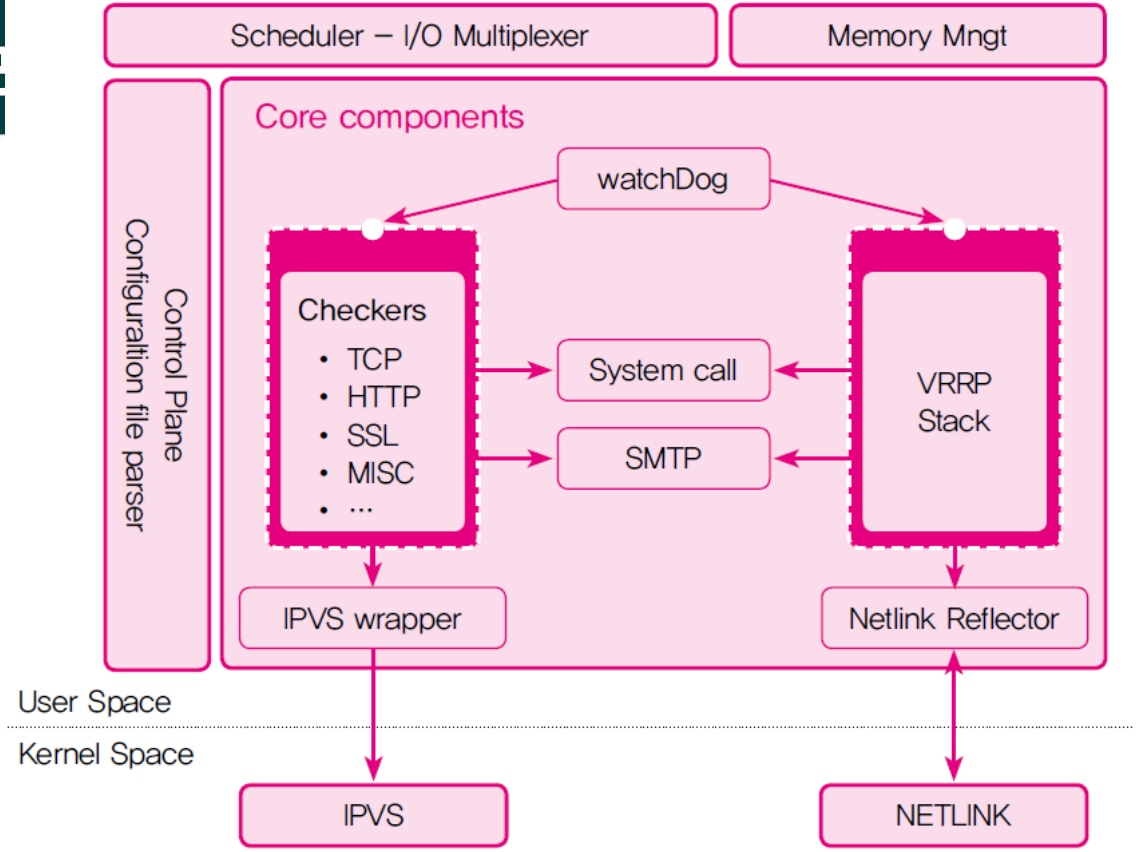
Steve McConnell 저

6.2 설계 원리(1)

- 시스템을 개발하는 조건이나 운용될 환경 조건의 제약 안에서 가능한 여러 설계 중에서 최적의 설계 안을 발견 하는 것
  - 설계를 평가하기 위한 특성과 기준을 명시하여야 한다. (정량적)
  - 정확성
  - 효율성(Efficiency)
    - 시스템이 사용하는 자원이 적정하고 효과적임을 의미한다.
  - 단순성(Simplicity)
    - 이해하기 쉬운 설계를 작성하는 것

- 소프트웨어 설계의 중심이 되는 원리
  - 단순화
  - 효율적
  - 단계적 분해(Stepwise refinement)
  - 추상화(Abstraction)
  - 모듈화(Modularization)

6.2 설계 원리

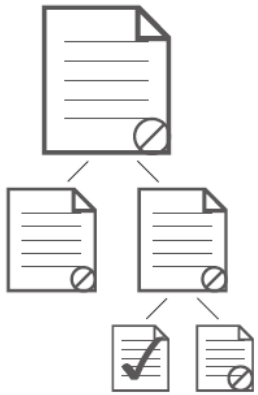


단순성	효율성	분할, 계층화	추상화	모듈화
복잡한 여러 가지 요소를 교통 정리하여 단순화 하거나 복잡함을 최소화 한다.	사용하는 자원이 적정하고 효과적이도록 한다.	다루기 쉬운 덩어리로 분리하여 계층화 한다.	자세한 부분에 좌우되지 않게 컴포넌트를 정의한다.	각 모듈이 외부와의 결합이 낮고 내부 요소가 응집되도록 한다.

단계적 분할

- 단계적 분해 ← divide-and-conquer
  1) 문제를 기본 단위로 나눈다.
  2) 독립된 문제로 구별한다.
  3) 구분된 문제의 자세한 내용은 가능한 한 뒤로 미룬다.
  4) 구체화 작업이 계속 점증적으로 일어난다는 것을 보인다. ➔ stepwise refinement
  * 자세한 것 (알고리즘, 자료구조)은 가급적 뒤로 미룬다

- 문제의 분할 개념



Declare and initialize variables
Input grades (prompt user and allow input)
Compute class average and output result

Compute:
add the grades
count the grades
divide the sum by the count

추상화(Abstraction)

- 정의

  필요한 부분만을 표현할 *수 있고 불필요한 부분을 제거하여 간결하고 쉽게 만드는 작업.*

- 추상화 방법
  - 기능 추상화

    입력자료를 출력자료로 변환하는 과정을 추상화

    부프로그램의 시그너쳐와 기능만 생각

  - 자료 추상화

    자료와 기능을 묶어서 생각

  - 제어 (control flow) 추상화

    외부 이벤트에 대한 반응을 추상화 (ex: 복합상태)

    switch문, 함수호출

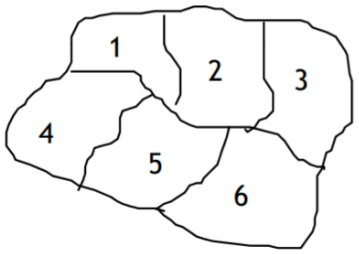
정보 은닉

- 모듈의 자세한 부분이 시스템의 다른 부분에서 감추어 있는 것
- 한 모듈이 다른 모듈에 구애 받지 않고 설계
- 인터페이스가 모듈 안의 구제척인 사항을 최소로 반영
  - 전역변수가 없어야
  - 모듈의 입력이 1이면 입력, 2이면 출력, ... ➔ 좋은 모듈 설계가 아님 ➔ 모듈화와 연관
- 모듈 단위의 수정, 시험, 유지보수에 장점
  - 모듈을 독립적으로 시험할 수 있으며, 모듈 별로 개선 및 최적화 할 수 있음

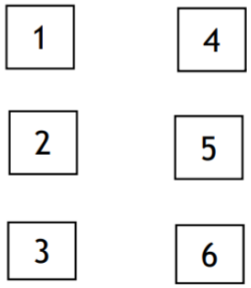
모듈화(1)

- **시스템의 분할을 어떻게?**

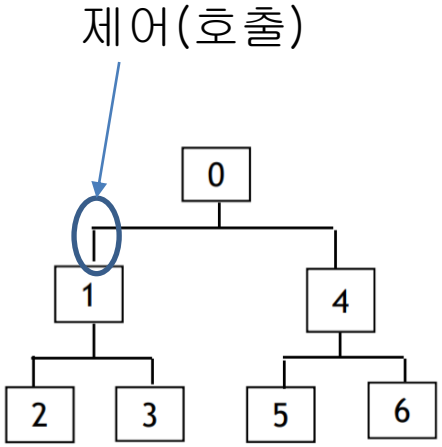
- 설계 과정에 독립된 모듈들로 분할하기 위하여 모듈 선택에 사용하는 기준이 있다. 기능 추상화 기법을 사용하는 시스템에서는 결합(coupling)과 응집(cohesion)이라는 두 가지 관점의 판단 기준이 적용된다.
- 기타 기준: 모듈 규모, 이식성



문제영역

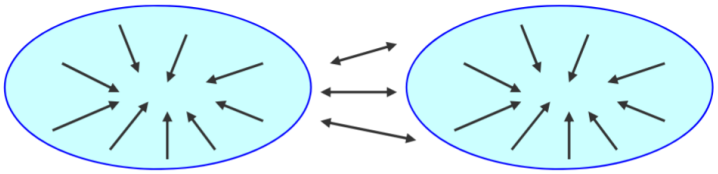


시스템 분해



구조도 (structure chart)

모듈화(2)



- 모듈 간의 결합(coupling)

(1) 자료 결합(data coupling)

모듈 간의 인터페이스가 자료 요소(파라미터)로만 구성된 경우 예) add(3, 5), sort(a)

(2) 스탬프 결합(stamp coupling)

모듈 간의 인터페이스를 통해 배열, 레코드가 전달되는 경우 (단, 배열, 레코드 내용 전체가 사용되면 “자료 결합”으로 볼 수 있음)
예) print_salary(인사 기록 레코드)

(3) 제어 결합(control coupling)

(4) 공통 결합(common coupling)

(5) 내용 결합(content coupling)

결합도 약함

결합도 강함

모듈화(2)

- 모듈 간의 결합(coupling)

(3) 제어 결합(control coupling)

한 모듈이 다른 모듈에게 제어 요소(function code, switch, flag 등)를 전달하는 경우 예) integer_operation('+', 3, 5)

(4) 공통 결합(common coupling)

공통된 자료 영역을 사용하는 경우 ➔ 자료 영역의 보호가 어렵고, 자료 구조 변경 시 파급 효과가 큼

예) C/C++ 등에서 global 변수를 사용하는 경우,
여러 process들이 공유하는 Shared Memory

(5) 내용 결합(content coupling)

한 모듈이 다른 모듈의 일부분을 직접 참조 또는 수정하는 경우 ➔ 공유 부분을 변경할 필요가 생기면 그 파급 효과가 큼

예 1) 에셈블리어에서 한 모듈이 다른 모듈의 데이터를 참조하는 경우

예 2) 한 모듈에서 다른 모듈로 분기(goto)하는 경우

모듈화(3)

- 모듈의 응집력(Cohesion)
  - 하나의 모듈은 전체 시스템이 갖는 여러 기능 중에 하나의 기능을 갖도록 설계해야...
  - 모듈 내의 모든 요소는 하나의 목적을 가지고 있는 것이 바람직하다.

모듈화(3)

- 모듈의 응집(cohesion)

(1) 기능적 응집(functional cohesion)

잘 정의된 하나의 기능이 하나의 모듈을 이룬 경우

예) "판매 세금 계산" ➔ (동사, 목적어) 한 쌍으로 구성

(2) 순차적 응집(sequential cohesion)

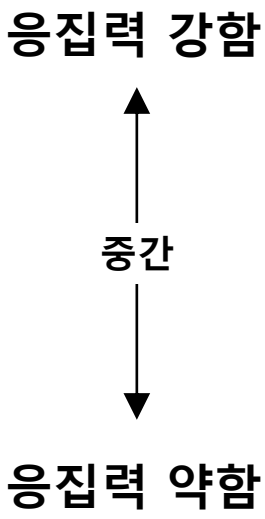
모듈 내 한 작업의 출력이 다른 작업의 입력이 되는 경우

예) "다음 거래를 읽고, 그 결과를 마스터 파일에 반영함"

(3) 교환적 응집(communication cohesion)

동일한 입력과 출력을 사용하는 여러 작업들이 모인 경우

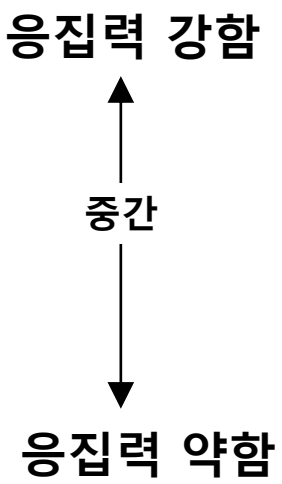
예) "인사 기록 파일에, 근무 성적을 기재하고 급여를 갱신함"



모듈화(3)

(4) 절차적 응집(procedural cohesion)

공유하는 것은 없으나, 큰 테두리 안에서 같은 작업에 속하는 경우 예) "총계를 출력하고, 화면을 지운다.



(5) 시간적 응집(temporal cohesion)

특정 시간에만 수행되는 기능을 묶어놓은 모듈

예) 초기화 루틴(변수 할당, 초기값 설정, ...), 타임아웃 핸들러

(6) 논리적 응집(logical cohesion)

유사 성격을 갖거나 특정 형태로 분류되는 처리 요소들을 한 모듈로 형성

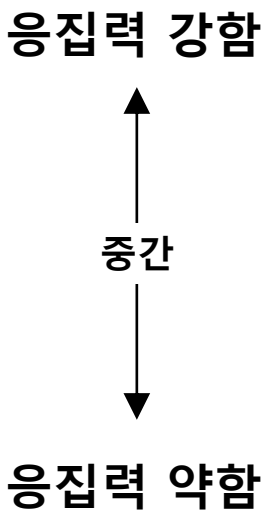
예) "사칙연산에서 주어진 매개변수에 따라 다른 계산을 함" ➔ 연산간 관계가 없음

(7) 우연적 응집(coincidental cohesion)

아무 관련 없는 처리 요소들로 모듈이 형성된 경우 ➔ 모듈 개념이 상실되어 이해 및 유지보수가 힘든 단점이 있음

모듈화(3)

- 모듈의 응집(cohesion)
  - (1) 기능적 응집(functional cohesion)
  - (2) 순차적 응집(sequential cohesion)
  - (3) 교환적 응집(communication cohesion)
  - (4) 절차적 응집(procedural cohesion)
  - (5) 시간적 응집(temporal cohesion)
  - (6) 논리적 응집(logical cohesion)
  - (7) 우연적 응집(coincidental cohesion)



모듈 기능을 정의한 문장	응집력
복문, 쉼표 하나 이상의 동사	순차적이나 교환적 응집
시간과 관련된 단어(처음, 다음, 이후 등)	순차적이나 시간적 응집
동사 앞에 단일 특정 객체가 아닌 경우	논리적 응집
초기화, 모두 삭제	시간적 응집

예제

Q1) 어떤 결합?

```
struct {
    int x, y;
} A;

int dolt(struct A p) {
   return p.x + p.y;
}
```

Q2) 어떤 결합 ?

```
void abc(void) {
     int a, b, in;     int res;
     scanf("%d %d %d", &a,
&b, &in);
    res = def(a, b, in);
    printf("result = %d", res);
}
```

```
int def(int x, int y, int v) {
   if(y > 0) {
      return (x+y);     }
   else {
      return (x*y);     }
}
```

Q3) 어떤 응집도 ?

한 모듈내에서 다수의 기능 요소들이 배열된 순서대로 수행되지만, 한 기능 요소에서 다른 기능 요소로 데이터(data) 가 아닌 제어(control)이 넘어가는 경우

Q4) 어떤 응집도?

A set of print functions generating different output reports from the same input data are arranged into a single module.

Q5) 어떤 응집도?

```
code = 8
new_operation(code, int a, int b, int c)
//parameters a, b & c are not used
// if code = 8.
```