6주 1강

소프트웨어 개발 방법

이번 주차에는…

소프트웨어 상위 설계

- 소프트웨어 개발 방법과 설계
- 객체지향의 주요 개념과 특징

1. 방법론이 만들어지는 과정

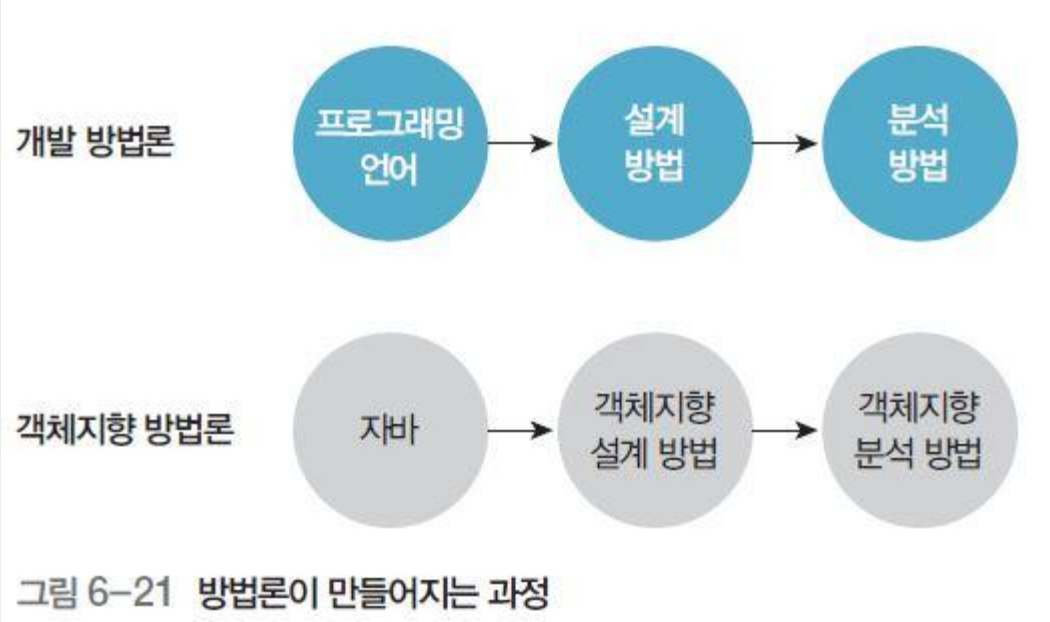


그림 6-21 방법론이 만들어지는 과정

2. 프로세스 지향 방법(1)

- procedural approach process oriented approach
  - 처리 순서를 구조화하는 방법
  - 대표적인 모델 기법: DFD(Data Flow Diagram)

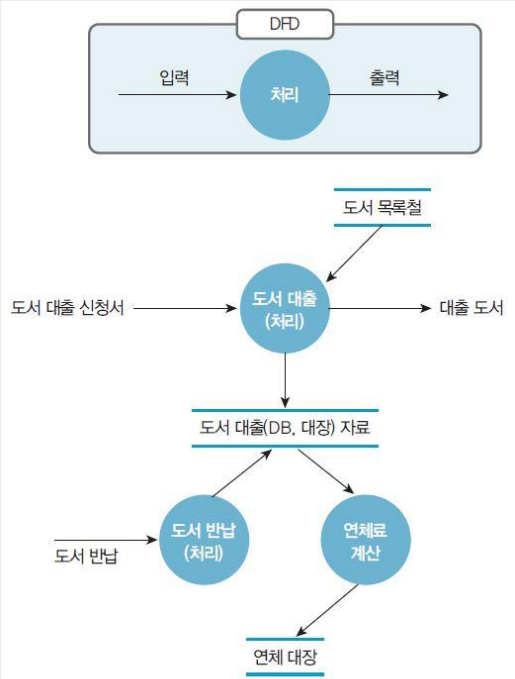


그림 6-22 DFD의 예

3. 프로세스 지향 방법(2)

- 프로세스 지향 방법의 구성
  - 기능이 중심(우선)이 되고 그 기능을 수행하는 데 필요한 데이터가 참조되는 형태로 구성

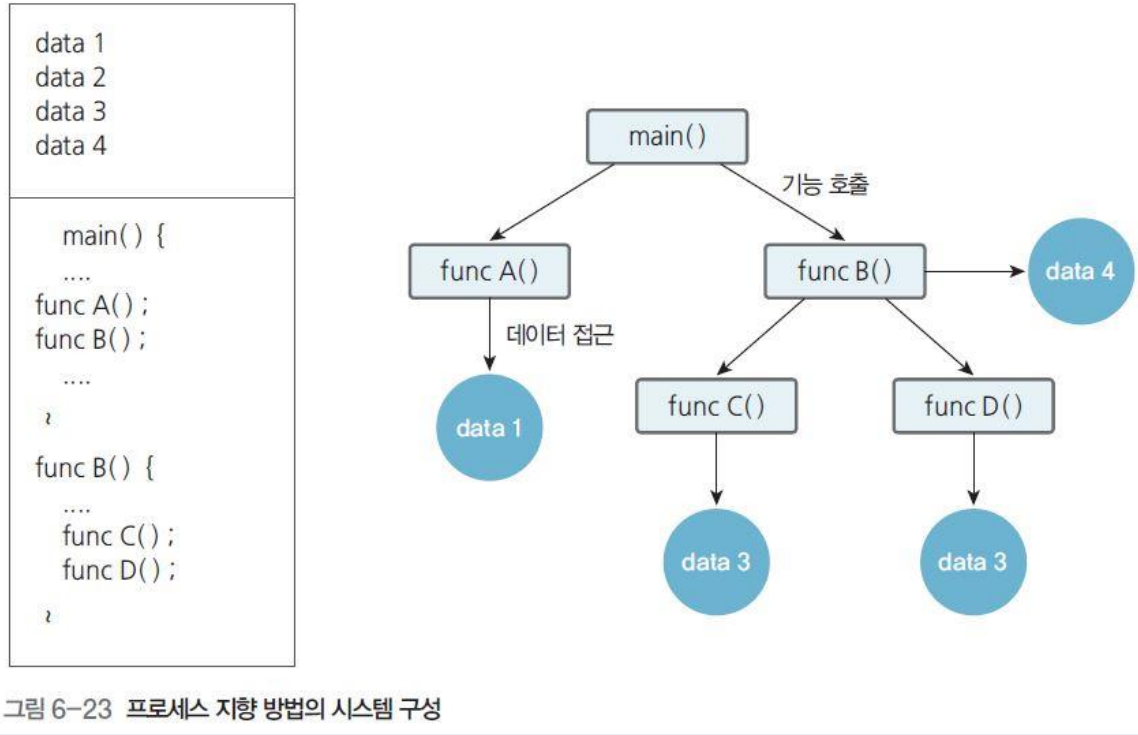


그림 6-23 프로세스 지향 방법의 시스템 구성

4. 프로세스 지향 방법(3)

- 프로세스 지향 방법의 특징
  - 프로세스와 데이터의 분리
  - 실세계를 컴퓨터 처리 방식으로 표현
  - 함수 중심(우선)으로 모듈 구성

5. 데이터 지향 방법

- 데이터 지향 방법data oriented approach
  - 시스템이 취급하는 데이터에 관심, 즉 데이터가 중심(우선)이 되어 데이터를 구조화
  - 대표적 소프트웨어 개발 방법론: 정보공학 방법론
  - DB 설계를 위한 대표적 모델 표기법: E-REntity-Relationship 다이어그램

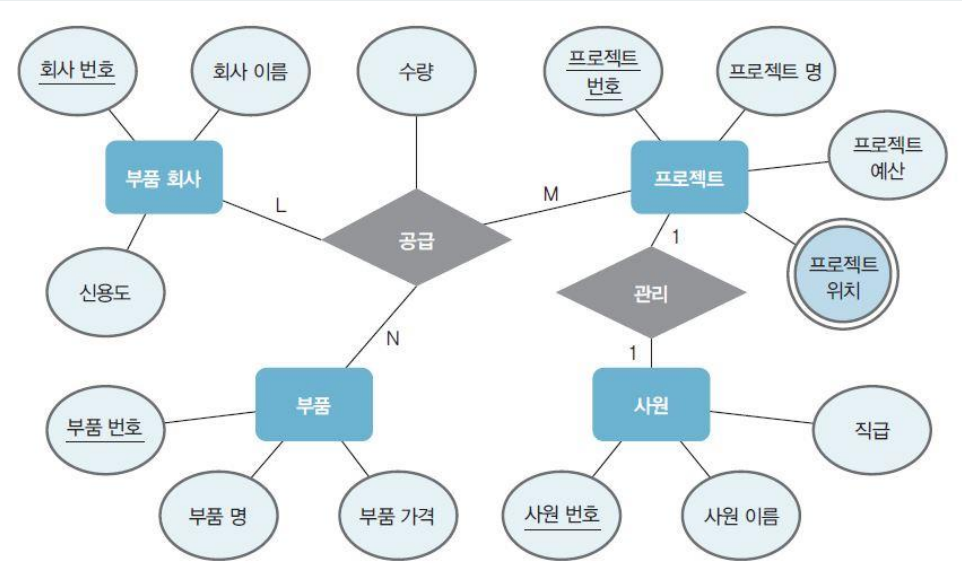


그림 6-24 E-R 다이어그램의 예

6. 프로세스 지향 방법과 데이터 지향 방법의 문제점

- 변경이 미치는 영향이 큼
  - 프로세스와 데이터를 각각 별개의 것으로 파악하기 때문
- 프로그램의 복잡도 증가
  - 함수와 데이터가 분리되어 있기 때문
- 프로그램 변경 시 프로그램 구조 파악 필요
  - 프로그래머는 프로그램의 구조와 영향을 미치는 곳도 파악해야 함
- 재사용의 어려움
  - 프로세스와 데이터가 분리된 구조 때문

7. 객체 지향 방법(1)

- **객체지향 방법**object-oriented approach
  - 프로세스 지향 방법과 데이터 지향 방법의 문제점을 해결하기 위해 고안
  - 기능이나 데이터 대신 객체가 중심이 되어 개발
  - 데이터(속성)를 가장 먼저 찾고 그 데이터를 조작하는 메서드(함수)를 찾아 그 둘을 객체라는 이름으로 묶어 그 객체를 중심으로 모듈을 구성

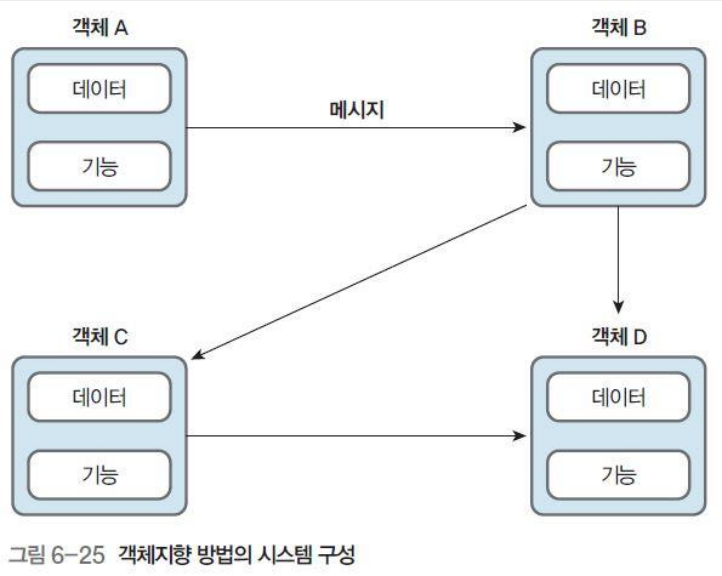


그림 6-25 객체지향 방법의 시스템 구성

8. 객체 지향 방법(2)

- 객체지향 방법의 특징
  - 실세계를 사람이 생각하는 방식으로 표현한다.
  - 임의로 데이터에 접근할 수 없다.
  - 시스템은 객체들의 모임이다
  - 요구 사항 변경에 유연하게 대처할 수 있다
  - 확장성과 재사용성이 높아진다.
  - 추상화를 통해 생산성과 품질이 높아진다

9. 객체(1)

표 6-1 객체지향의 주요 개념과 특징

주요 개념	특징
• 객체 • 클래스 • 인스턴스	• 캡슐화 • 정보은닉 • 상속 • 다형성

10. 객체(2)

- 객체object
  - 실세계에 존재하거나 생각할 수 있는 것들
  - 사전에 나와 있는 명사뿐 아니라 동사의 명사형까지도 모두 객체
  - 인간이 생각하고 표현할 수 있는 모든 것

(a) 실세계의 객체 (b) 소프트웨어 개발의 객체

그림 6-26 객체

11. 객체(3)

- 관점에 따른 객체의 이해
  - 모델링 관점 : 객체는 명확한 의미를 담고 있는 대상 또는 개념
  - 프로그래머 관점 : 객체는 클래스에서 생성된 변수
  - 소프트웨어 개발 관점 : 객체는 '데이터+메서드' 형태의 소프트웨어 모듈
  - 객체지향 프로그래밍 관점 : 객체는 속성attribute과 메서드method 용어로 구현

- 개발 관점에서의 객체의 특성
  - 식별자identity 존재 : 객체를 구별하는 유일한 식별자를 갖는다.
  - 상태state 존재 : 자료구조에 해당하는 상태를 갖는다.
  - 메서드 존재 : 연산을 수행할 수 있는 행위에 해당하는, 잘 정의된 메서드를 갖는다.
  - 클래스로 선언 및 사용 : 객체는 비슷한 객체의 구조와 행위가 클래스로 선언되어 사용

12. 객체지향 프로그래밍(1)

- OOP(Object-Oriented Programming)는 객체의 관점에서 프로그래밍을 하는 것을 의미
- 절차 지향 프로그래밍
  - 프로세스가 함수 단위로 순서대로 진행
- 객체지향 프로그래밍
  - 객체의 유기적인 관계를 통해 프로세스가 진행
- 추상화
  - 목적과 관련 없는 부분은 제거하고 필요한 부분만 표현
  - 객체들의 공통된 특징을 파악해 정의한 설계 기법
- 캡슐화
  - 불필요한 정보를 감추는것
  - 정보은익과 다른 의미
- 상속
  - 부모클래스에서 자식클래스에게 물려주는것
  - 포유류 클래스 → 강아지 클래스, 고양이 클래스에게 공통 부분 물려줌

13. 객체지향 프로그래밍(2)

- 다형성
  - 형태가 같은데 다른 기능을 수행한느것
  - 부모 클래스에서 상속 받은 속성(메소드)가 자식 클래스에 따라 다른 형태로 표현되는것
- 오버라이딩
  - 부모 클래스에서 상속 받은 속성을 자식 클래스에서 재 정의하여 사용하는 것
- 인스턴스
  - 설계도를 바탕으로 소프트웨어에서 실제적으로 구현된 구체적인 실체
  - 객체를 소프트웨어에서 실체화 하면 그것을 “인스턴스”라고 함
  - 인스턴스는 어떠한 원본으로부터 ‘생성된 복제품’이라 할 수 있다.
- 클래스 & 객체 & 인스턴스
  - 클래스는 설계도, 객체는 설계도를 바탕으로 구현된 모든 대상
  - 인스턴스는 객체가 메모리에 할당되어 실제 사용 될 때 인스턴스라 함

14. 클래스(1)

- 클래스
  - 클래스class는 공통되는 것들을 묶어서 대표적인 이름을 붙인 것
  - 클래스가 개념적이라면, 객체는 구체적
  - 데이터뿐 아니라 이 데이터에서 수행되는 메서드까지 포함해서 묶어놓은 것

클래스	승용차
객체	소나타, 그랜저, SM5

클래스	자동차
객체	승용차, 버스, 트럭

클래스	운송 수단
객체	자동차, 배, 비행기

그림 6-27 클래스와 객체의 예

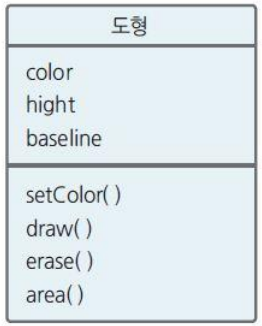
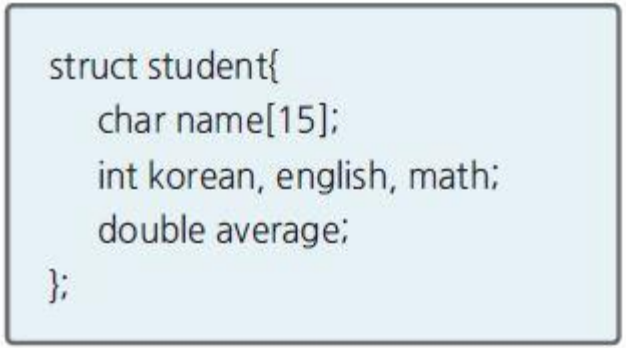


그림 6-29 클래스의 예

15. 클래스(2)

- 구조체
  - 서로 연관된 자료들만 모아놓은 것



```
struct student{
    char name[15];
    int korean, english, math;
    double average;
};
```

그림 6-28 구조체의 예

[구조체의 구성]

- struct : 구조체를 나타내는 예약어
- student : 구조체 태그명
- 구조체 멤버 : name, korean, english, math, average

16. 인스턴스

- 인스턴스instance
  - 같은 클래스에 속하는 개개의 객체로, 하나의 클래스에서 생성된 객체
  - 클래스가 구체화되어, 클래스에서 정의된 속성과 성질을 가진 실제적인 객체로 표현된 것
  - 인스턴스화instantiation : 추상적인 개념인 클래스에서 실제 객체를 생성하는 것

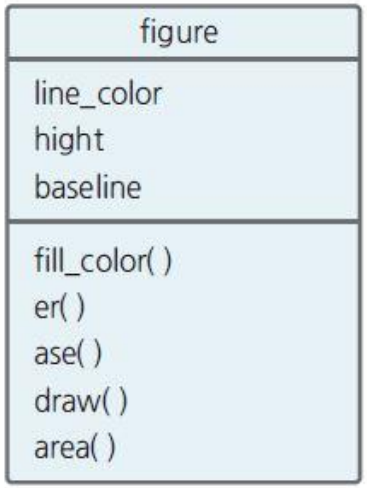
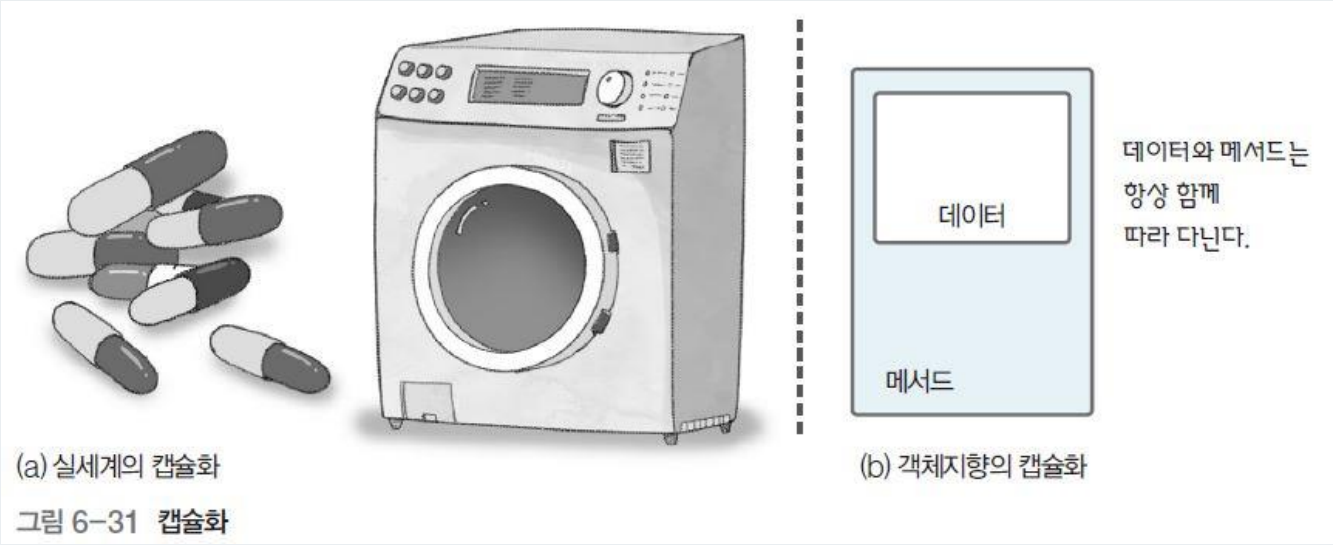


그림 6-30 인스턴스의 예

17. 캡슐화(1)

- **캡슐화**encapsulation
  - 사용자들에게 해당 객체의 기능(서비스)과 사용법만 제공하고 내부는 감추어(변경할 수 없게 함) 쉽게 사용할 수 있게 하는 개념
  - 객체 내부에 서로 관련된 데이터와 그 데이터를 조작할 수 있는 메서드를 같이 포장하는 방식으로 그 안에 포함된 메서드만 사용하여 데이터 값을 변경할 수 있는 구조



(a) 실세계의 캡슐화 (b) 객체지향의 캡슐화

그림 6-31 캡슐화

18. 캡슐화(2)

- 캡슐화의 장점
  - 데이터 보호
  - 추상화 용이
  - 제공자와 이용자를 명확히 분리
  - 이용자에게 편리성 제공
  - 사용법이 쉬움
  - 변화에 대한 국지적 영향
  - 객체 간의 독립성 보장
  - 변경 용이성과 재사용성 증대

19. 정보은닉(1)

그림 6-32 정보은닉의 일상 예

20. 정보은닉(2)

- 정보은닉information hiding
  - 외부(다른 객체)에서 객체의 내부(데이터)를 들여다볼 수 없다는 개념
  - 다른 객체가 한 객체 내의 데이터 값을 직접 참조하거나 접근할 수 없는 구조
  - 인터페이스와 구현의 명확한 분리
  - 각 모듈의 내부 항목에 관한 정보는 감추고, 인터페이스를 통해서만 메시지를 전달
  - 다른 모듈을 변경하지 못함
  - 모듈 안의 자료구조와 메서드에 사용된 알고리즘은 외부에서 그 값을 직접 변경 못함
  - 공개 인터페이스로 정의 된 메서드를 통해서만 접근 가능

21. 정보은닉(3)

- 정보은닉의 표기 방법

표 6-2 프로그래밍 언어별 정보은닉의 표기 방법

	단계	C++	자바
공개	+	public	public
부분 공개	#	protected	protected
은닉	–	private	private

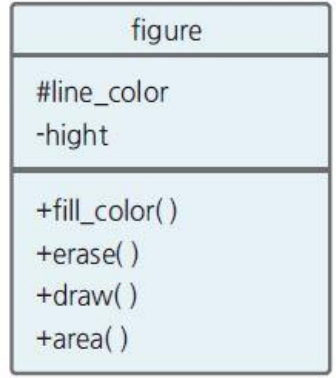


그림 6-33 정보은닉의 표기

22. 정보은닉(4)

- 정보은닉의 특징
  - 블랙박스 역할
  - 인터페이스를 통한 접근
  - 자료구조 변경이 용이

- 정보은닉 개념 사용의 장점
  - 독립성 향상
  - 수정 용이
  - 이해도 증진
  - 확장성 증가

23. 상속(1)

- 상속inheritance
  - 뭔가를 물려받는다는 의미
  - 상위 클래스super class의 모든 것을 하위 클래스sub class가 물려받아 내 것처럼 사용

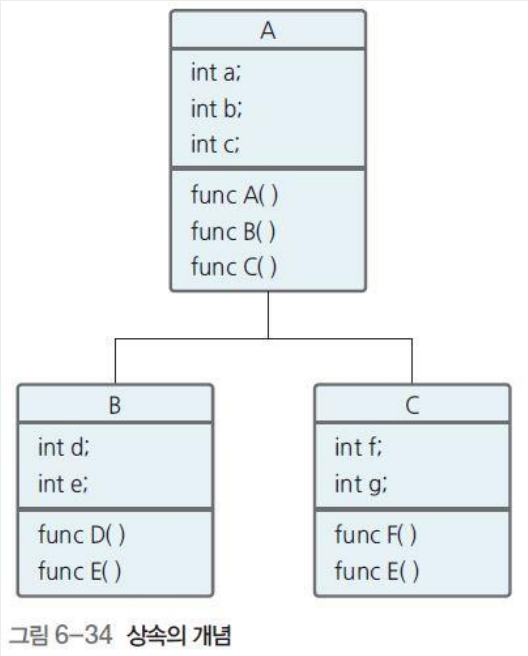


그림 6-34 상속의 개념

24. 상속(2)

- 상속의 장점
  - 이해 용이
  - 재사용성 증대
  - 확장 용이
  - 유지보수 용이
  - 추상화 가능

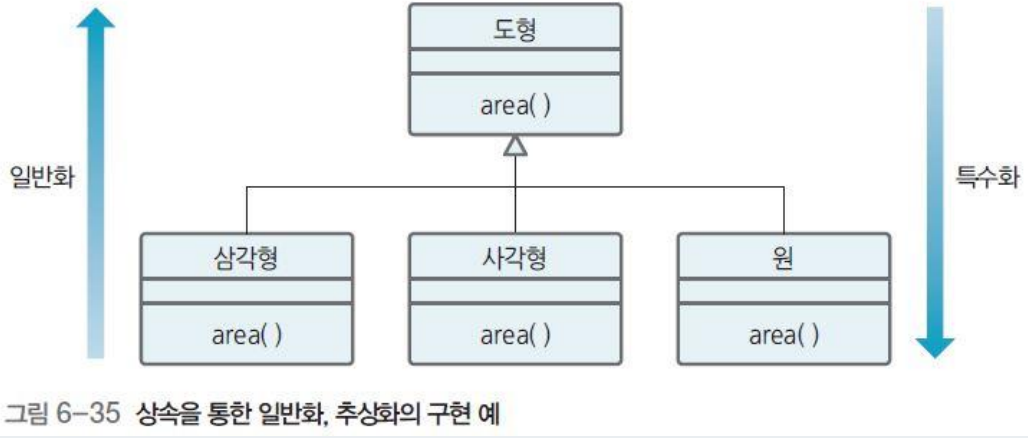


그림 6-35 상속을 통한 일반화, 추상화의 구현 예

25. 다형성(1)

- 다형성polymorphism
  - '여러 개의 형태를 갖는다'라는 의미의 그리스어에서 유래
  - poly(하나 이상), morph(형태)가 합성된 단어로 '하나 이상의 형태'를 뜻함

- 다형성 예 1
  - 다음 그림에서 공통점: '타다'

그림 6-36 다형성의 예 : 탈 것

26. 다형성(2)

- **다형성 예 2**
  - C 언어에서 + 기호는 다음 두 가지 용도로 사용된다.
    - 연산자operator : 두 수를 더하는 연산자로, '3+5' 형태로 사용
    - 연결자concatenation : 문자열을 연결하는 역할을 하며 'go+stop' 형태로 사용

- **다형성 예 3**
  - **'삼각형 면적을 계산한다', '사각형 면적을 계산한다', '원 면적을 계산한다'의 공통점? '면적을 계산한다'**

- **다형성 예 4**
  - **'창문을 열다', '지갑을 열다', '파일을 열다', '은행 계좌를 열다'의 공통점: '열다'**

27. 다형성(3)

- 다형성
  - 같은 이름의 메서드가 객체에 따라 다르게 동작하고, 서로 다른 구현(코드)을 제공

도형	면적을 구하는 메서드	면적을 구하는 공식
삼각형	area( )	밑변×높이÷2
사각형	area( )	가로×세로
원	area( )	반지름^2×3.14

그림 6-38 도형에 따라 달라지는 면적 공식

- 다형성 사용의 장점
  - 쉬운 변경(추가, 삭제), 확장 및 유지보수의 용이

그림 6-39 사다사리꼴을 추가하는 경우

28. 오버로딩 & 오버라이딩

- 오버로딩(OverLoading)
  - 클래스에서 같은 이름의 속성(메소드)를 여러 개 가지면서 매개변수의 유형과 개수에 따라 다르게 사용하는것
- 오버라이딩(Overriding)
  - 상위 클래스가 있는 속성(메소드)를 하위 클래스가 재정의하여 사용하는 것

구분	오버로딩	오버라이딩
메소드 이름	동일	동일
매개변수, 타입	다름	동일
리턴 타입	상관없음	동일

29. 메소드 오버로딩(1)

- 메서드 오버로딩overloading
  - 한 클래스에 이름이 동일한 메서드가 중복 정의되어 있는 경우
  - 메서드명이 같은데 어떻게 구별할까?
    → 매개변수 타입이나 개수signature로 구별

그림 6-40 메서드 오버로딩의 예

30. 메소드 오버로딩(2)

- 연산자 오버로딩
  - 연산자 하나를 다른 용도로 다시 중복 정의하여 사용하는 것
    (예) '3+5', 'go+stop'

- 상속 구조에서의 메서드 오버로딩
  - 상속 구조에서는 메서드명뿐아니라 매개변수의 타입과 개수까지 같다. 구별 방법?
    → 상위 클래스: 추상 클래스, 추상 클래스내의 메서드: 추상 메서드

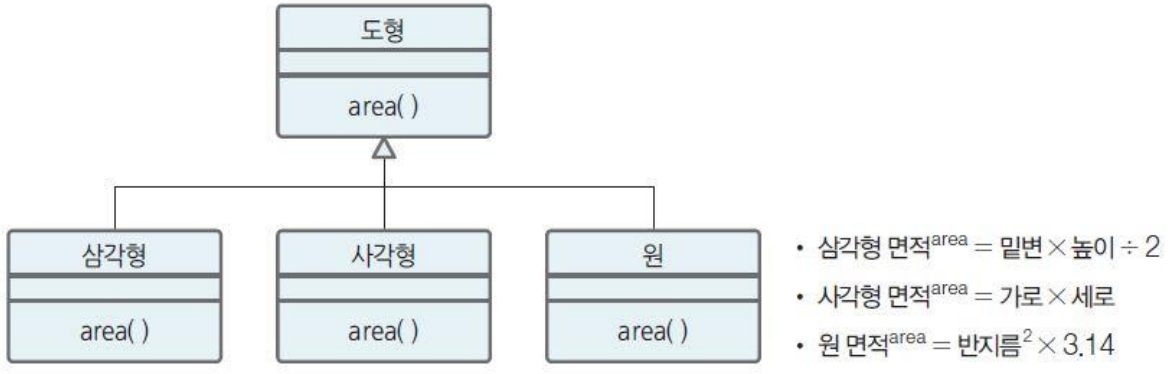


그림 6-41 상속 구조에서의 메서드 오버로딩

31. 메소드 오버라이딩

- 메서드 오버라이딩overriding
  - 메서드 오버로딩: 추상 클래스와 추상 메서드만 사용
  - 메서드 오버라이딩: 추상 클래스와 일반 클래스를 모두 다 사용
  - 상위 클래스에서 정의한 일반 메서드의 구현을 하위 클래스에서 모두 무시하고 다시 재정의해서 사용 가능 → 상위 클래스의 메서드는 은닉(무시)되고, 하위 클래스의 메서드가 사용

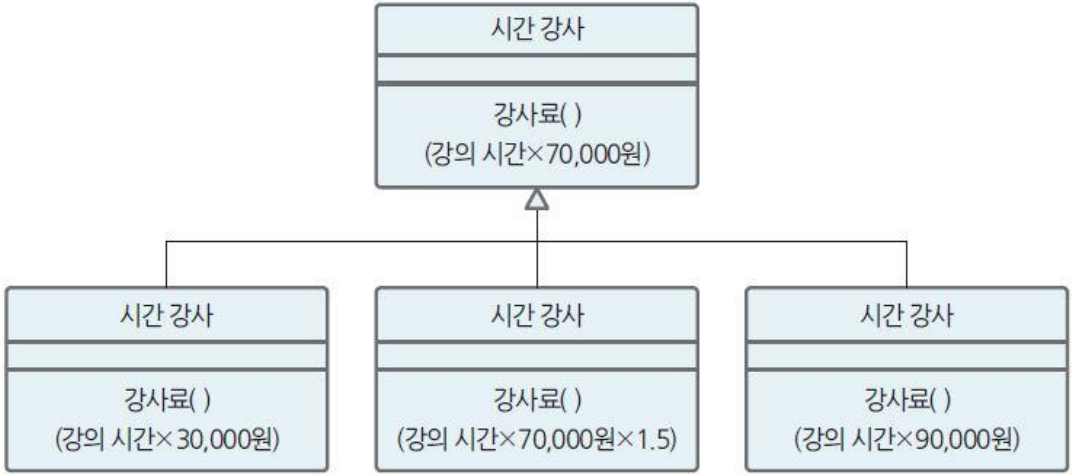


그림 6-42 메서드 오버라이딩의 예

다음 시간

클래스의 관계와 설계 원칙