6. 설계 및 구현

주요내용

- 프로젝트에서 설계란 무엇인가?
- 프로젝트에서 설계는 왜 중요한가?
- 프로젝트에서 설계 원리는 무엇인가?
- 효과적인 모듈 설계는 어떠해야 하는가?
- 객체지향 설계란 무엇인가?
- 구현 작업이란 무엇인가?

목차

- 강의 내용
  - 설계의 정의
  - 상위 설계와 하위 설계
  - 설계 원리
  - 효과적인 모듈 설계
  - 객체지향의 개념
  - 구현
- 팀 프로젝트 (10, 11주차)
  - 설계 문서 작성 및 제출

설계란?

- 정의
  - 설계는 개발될 제품에 대한 의미 있는 공학적 표현
  - 설계는 고객의 요구사항으로 추적 가능해야 하며, 동시에 좋은 설계라는 범주에 들도록 품질에 대해서도 검증되어야 한다
    [IEEE-Std-610]

- 소프트웨어의 설계(design)
  - 본격적인 프로그램의 구현에 들어가기 전에 소프트웨어를 구성하는 뼈대를 정의해 구현의 기반을 만드는 것
  - 종류
    - 상위 설계(High-Level Design)
    - 하위 설계(Low-Level Design)

상위 설계와 하위 설계

❖ 상위 설계(High-Level Design)

– 의미

- 아키텍처 설계(Architecture Design), 예비 설계(Preliminary Design)라고 함
- 시스템 수준에서의 소프트웨어 구성 컴포넌트들 간의 관계로 구성된 시스템의 전체적인 구조
- 시스템 구조도(Structure Chart), 외부 파일 및 DB 설계도(레코드 레이아웃, ERD), 화면 및 출력물 레이아웃 등이 포함됨

❖ 하위 설계(Low-Level Design)

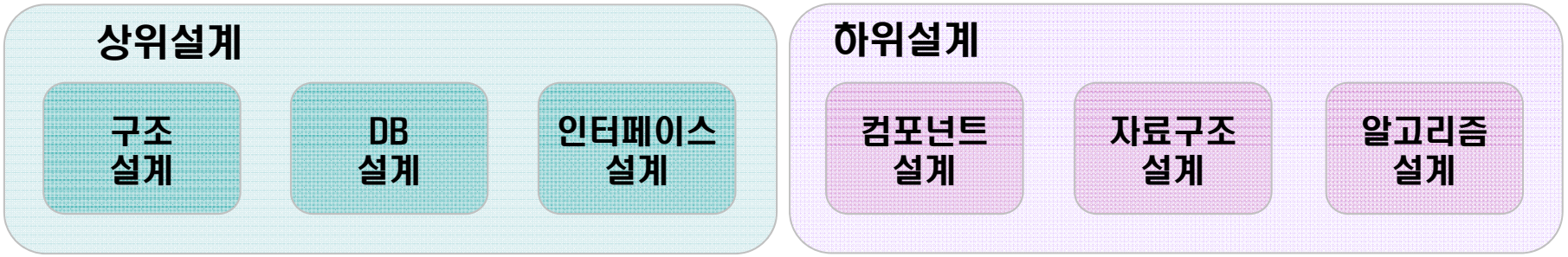
– 의미

- 모듈 설계(Module Design), 상세 설계 (Detail Design)이라고 함
- 시스템의 각 구성 요소들의 내부 구조, 동적 행위 등을 결정
- 각 구성 요소의 제어와 데이터들간의 연결에 대한 구체적인 정의를 하는 것

– 하위 설계 방법

- 절차기반(Procedure-Oriented), 자료위주(Data-Oriented), 객체지향(Object-Oriented) 설계 방법

상위 설계와 하위 설계의 구조도



설계 프로세스

설계 프로세스

- 좋은 설계란
  - 요구사항 명세서의 모든 내용을 구현해야 한다
  - 이해가 쉬워서 구현 또는 테스트로 추적이 가능해야 한다
  - 유지 보수 시 변경이 용이해야 한다

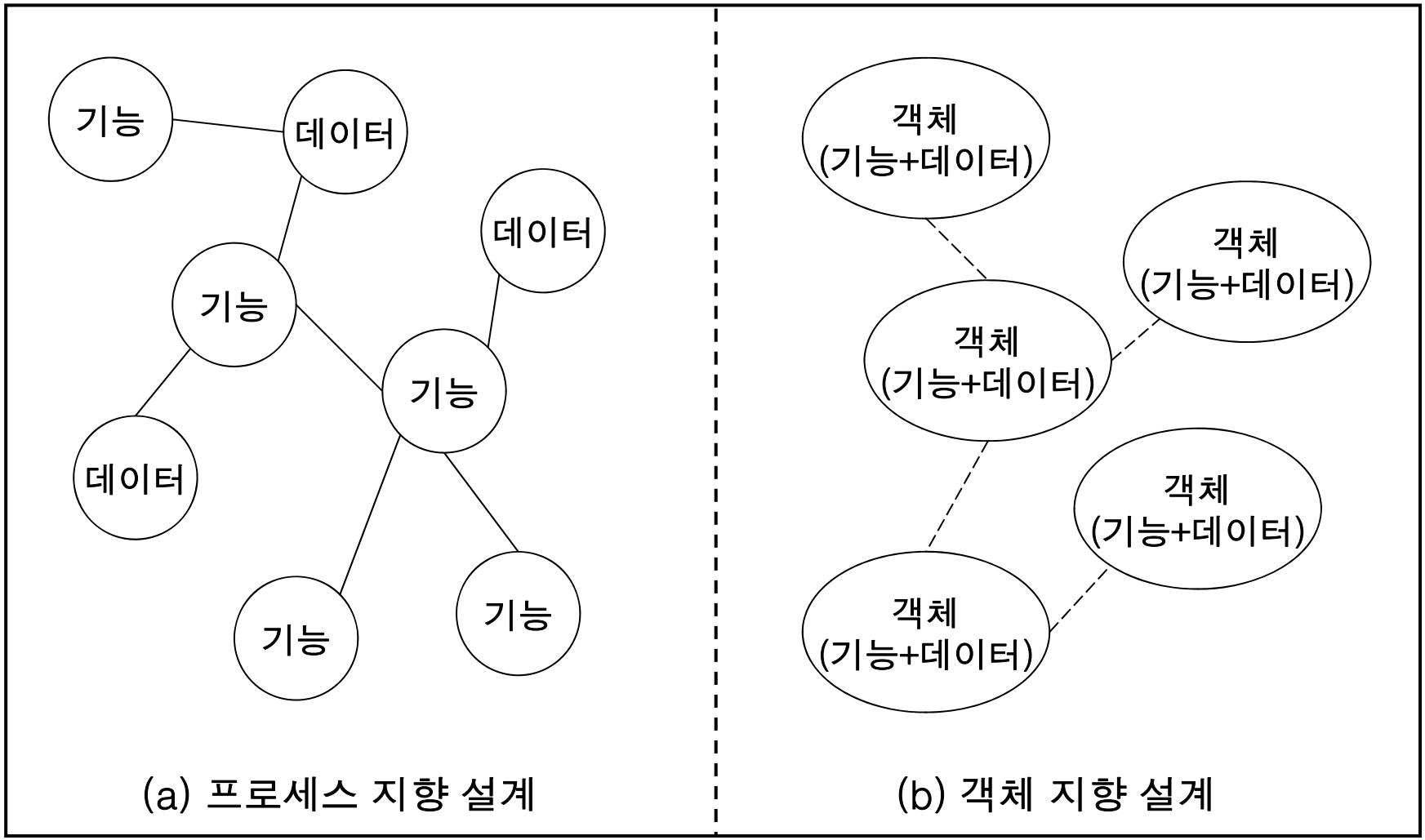
- 설계 방식
  - 프로세스 지향 설계(Process Oriented Design)
  - 객체지향 설계(Object Oriented Design)

설계 방식

- 프로세스 지향 설계(Process Oriented Design)
  - 업무의 처리절차를 중심으로 설계의 구성 요소들을 구분
  - 어떠한 절차를 거쳐서 작업을 수행하는가, 어떠한 입출력 자료를 생성하는가에 초점
  - 시스템은 “기능과 데이터”들이 노드를 이루고 이들의 관계가 링크를 형성하는 그래프

- 객체지향 설계(Object Oriented Design)
  - 시스템의 실제 객체 요소를 중심으로 설계
  - 자료구조와 그에 대한 연산을 묶어서 구성되는 객체들을 정의하고 이들이 상호 작용의 기본이 되도록 설계
  - 객체들이 노드를 이루고 이들간의 관계가 링크를 형성하는 그래프

시스템을 해석하는 관점의 차이



설계 원리

설계 원리

- 추상화(Abstraction)
- 단계적 분해(Stepwise refinement)
- 모듈화(Modularization)

추상화(Abstraction)

- 의미
  - 자세한 구현에 전에, 상위 레벨에서의 제품의 구현을 먼저 생각해보는 것
- 단계
  - 상위 레벨에서 설계를 생각해본 후 점차 구체적인 단계로 옮겨가는 것
- 종류
  - 과정 추상화(Procedure Abstraction)
  - 데이터 추상화(Data Abstraction)
  - 제어 추상화(Control Abstraction)

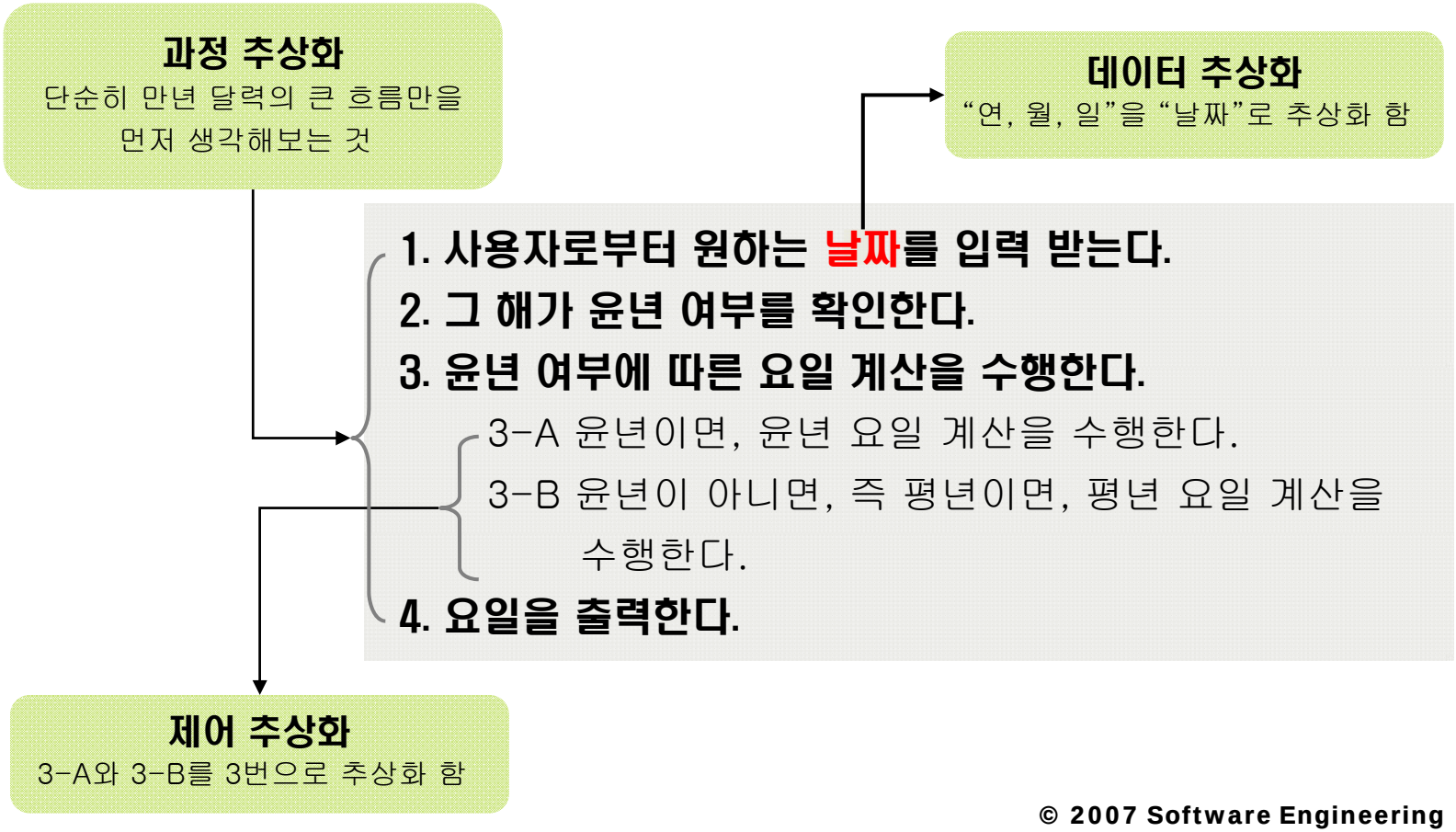
추상화의 종류 (1/2)

- **과정 추상화**
  - 수행 과정의 자세한 단계를 고려하지 않고, 상위 수준에서 수행 흐름만 먼저 설계
- **데이터 추상화**
  - 데이터 구조를 대표할 수 있는 표현으로 대체하는 것
  - 예) 날짜 구조를 단순히 "날짜" 로 추상화 하는 것
- **제어 추상화**
  - 3-A와 3-B를 "3. 윤년 여부에 따라 요일 계산을 수행한다."로 추상화 하는 것

추상화의 종류 (2/2)

❖ 예제

- **원하는 날짜를 입력으로 받아 요일을 알려주는 만년 달력 프로그램**



단계적 분해(Stepwise Refinement)

- 의미
  - Niklaus Wirth에 의해 제안됨
  - 문제를 상위 개념부터 더 구체적인 단계로 분할하는 하향식 기법의 원리
  - 모듈에 대한 구체 설계를 할 때 사용

- 과정
  - 문제를 하위 수준의 독립된 단위로 나눈다.
  - 구분된 문제의 자세한 내용은 가능한 한 뒤로 미룬다.
  - 점증적으로 구체화 작업을 계속한다.

모듈화

- 모듈의 의미
  - 수행 가능 명령어, 자료구조 또는 다른 모듈을 포함하고 있는 독립 단위
- 특성
  - 이름을 가지며
  - 독립적으로 컴파일 되고
  - 다른 모듈을 사용할 수 있고
  - 다른 프로그램에서 사용될 수 있다
- 모듈의 예
  - 완전한 독립 프로그램, 라이브러리 함수, 그래픽 함수 등
- 모듈의 크기
  - 되도록 쉽게 이해될 수 있도록 가능한 한 작아야 함
  - 너무 작은 모듈로 나눠지지 않도록 함

효과적인 모듈 설계

정보 은닉(Information Hiding)

- 의미
  - 각 모듈 내부 내용에 대해서는 비밀로 묶어두고, 인터페이스를 통해서만 메시지를 전달 할 수 있도록 하는 개념
  - 설계상의 결정 사항들이 각 모듈 안에 감추어져 다른 모듈이 접근하거나 변경하지 못하도록 함

- 장점
  - 모듈의 구현을 독립적으로 맡길 수 있음
  - 설계 과정에서 하나의 모듈이 변경되더라도 설계에 영향을 주지 않음

정보은닉의 예

```
void main( )
{
   stack st1;
   char x, y;

   create_stack(st1);
   push(st1, 'a')
   push(st1, 'b')

   x = pop(st1);
   y = top_element(st);


   destory_stack(st1);
   printf("%c, %c/n", x, y);
}
```

예제 A

```
void main( )
{
   stack* st1;
   char x, y;

   st1 = new stack;
   st1->top = 0;
   push(st1, 'a')
   push(st1, 'b')

   x = pop(st1);
   y = st1->stack_value[st1->top - 1];


   delete st1;
   printf("%c, %c/n", x, y);
}
```

예제 B

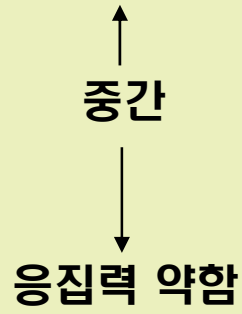
모듈의 응집력 (1/2)

❖ 모듈의 응집력이란?

- 모듈을 이루는 각 요소들의 서로 관련되어 있는 정도
- 강력한 응집력을 갖는 모듈을 만드는 것이 모듈 설계의 목표

❖ Myers의 응집력 정도 구분

1. 기능적 응집(Functional cohesion)
2. 교환적 응집(Communication cohesion)
3. 절차적 응집(Procedural cohesion)
4. 시간적 응집(Temporal cohesion)
5. 논리적 응집(Logical cohesion)
6. 우연적 응집(Coincidental cohesion)



모듈의 응집력 (2/2)

❖ 응집력의 종류

- **기능적 응집(Functional cohesion)**
  - 모듈이 잘 정의된 하나의 기능만을 수행할 때 기능적 응집도가 높아짐
- **교환적 응집(Communication cohesion)**
  - 동일한 입/출력을 사용하는 작은 작업들이 모인 모듈에서 볼 수 있음
- **절차적 응집(Procedural cohesion)**
  - 모듈 안의 작업들이 큰 테두리 안에서 같은 작업에 속하고, 입출력을 공유하지 않지만 순서에 따라 수행될 필요가 있는 경우
- **시간적 응집(Temporal cohesion)**
  - 프로그램의 초기화 모듈 같이 한 번만 수행되는 요소들이 포함된 형태
- **논리적 응집(Logical cohesion)**
  - 비슷한 성격을 갖거나 특정 형태로 분류되는 처리 요소
- **우연적 응집(Coincidental cohesion)**
  - 아무 관련 없는 처리 요소들로 모듈이 형성되는 경우

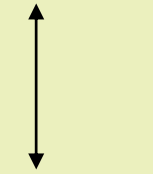
모듈의 결합도 (1/2)

❖ 의미

- 모듈간에 연결되어 상호 의존하는 정도
- 낮은 결합도를 갖는 모듈(Loosely coupled)을 만드는 것이 모듈 설계의 목표

❖ 모듈간의 의존도

자료 결합(Data coupling)
구조 결합(Stamp coupling)
제어 결합(Control coupling)
공통 결합(Common coupling)
내용 결합(Content coupling)



모듈의 결합도 (2/2)

❖ 결합도의 종류

- **자료 결합(data coupling)**
  - 모듈 간의 인터페이스가 자료 요소로만 구성된 경우
  - 가장 이상적인 형태의 결합
- **구조 결합(stamp coupling)**
  - 모듈 간의 인터페이스로 배열이나 레코드 등의 자료 구조가 전달되는 경우
- **제어 결합(control coupling)**
  - 한 모듈이 다른 모듈에게 제어 요소(function code, switch, tag 등)를 전달하는 경우
- **공통 결합(common coupling)**
  - 여러 모듈이 공동 자료 영역을 사용하는 경우
- **내용 결합(content coupling)**
  - 한 모듈이 다른 모듈의 일부분을 직접 참조 또는 수정하는 경우

객체지향(Object-Oriented) 개념

객체지향

❖ 객체지향의 등장 배경

- 기존의 구조적 기법으로 유지보수가 어렵다는 단점을 극복하기 위해 등장

❖ 객체란?

- 특성(Attribute)와 행위(Behavior)를 가지고 있는 인지할 수 있는 개체(Entity)
  - 특성
    - ➢ 해당 객체에 저장되어 있는 데이터
  - 행위
    - ➢ 객체가 할 수 있는 일, 객체의 상태가 변하게 하는 원인을 제공
- 다른 객체와 구별할 수 있는 정체성(identity)을 가짐
  - 정체성
    - ➢ 해당 객체를 다른 개체와 구별 할 수 있는 식별 값

❖ 객체의 예: 차

- 특성: 검정색 차체, 6기통 엔진, 자동 변속기, 4개의 바퀴 등
- 행위: 출발하다, 정지하다, 가속하다, 감속하다 등
- 정체정: 차량 번호

클래스와 객체

❖ 클래스(Class)

- 여러 객체들을 위한 대표적 구조
- 객체들이 내부적으로 어떻게 구성되어 있는지 설명
- 객체의 특성은 클래스의 변수로, 행위는 메소드로 표현됨

❖ 객체(Object)

- 클래스의 실례 또는 실체(Instance)라고도 부름
  - 객체가 클래스에서 정의하는 변수, 메소드를 그대로 가지면서 메모리에 할당되기 때문
- 각 객체 내부의 변수 이름은 같지만 서로 독립적임

```
public class Car
{
    ...
    String carName; //차 이름
    Int carNumber; //차 번호
    ...
}
Car myCar = new Car();
```

클래스

객체

객체지향 방법의 특징

- 절차를 강조하는 구조적 방법
  - 데이터를 소홀히 하게 됨

- 객체지향 방법
  - 시스템을 구성하는 요소들은 객체로,
  - 시스템 개발의 복잡한 문제들을 캡슐화(Encapsulation), 상속(Inheritance), 다형성(Polymorphism) 개념으로 해결하려 함

캡슐화(Encapsulation) (1/2)

- 의미
  - 소프트웨어 모듈인 객체의 내부에 가진 상세한 정보와 처리 방식을 외부로부터 감추는 것
  - 객체의 추상화를 통해 독립성을 보장해 주는 개념

캡슐화(Encapsulation) (2/2)

❖ 캡슐화의 예

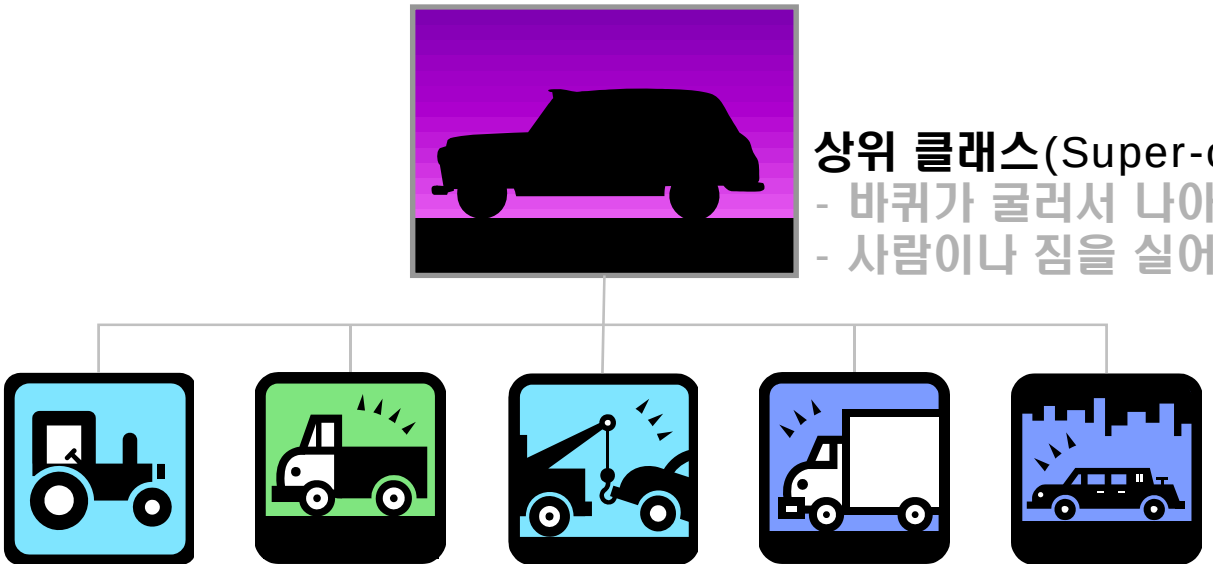
```
class Car
{
    ...
      private String carName; //차 이름
      private Int carNumber; //차 번호

      public String getCarName() // 차 이름 반환
      {
            return carName;
      }

      public Int getCarNumber() // 차 번호 반환
      {
            return carNumber;
      }
    ...
}
```

private 이용
외부에 감춤

public 이용
외부에 공개

상속(Inheritance) (1/2)

- 의미
  - 다른 클래스의 속성을 물려받아 내 것처럼 쓰는 것

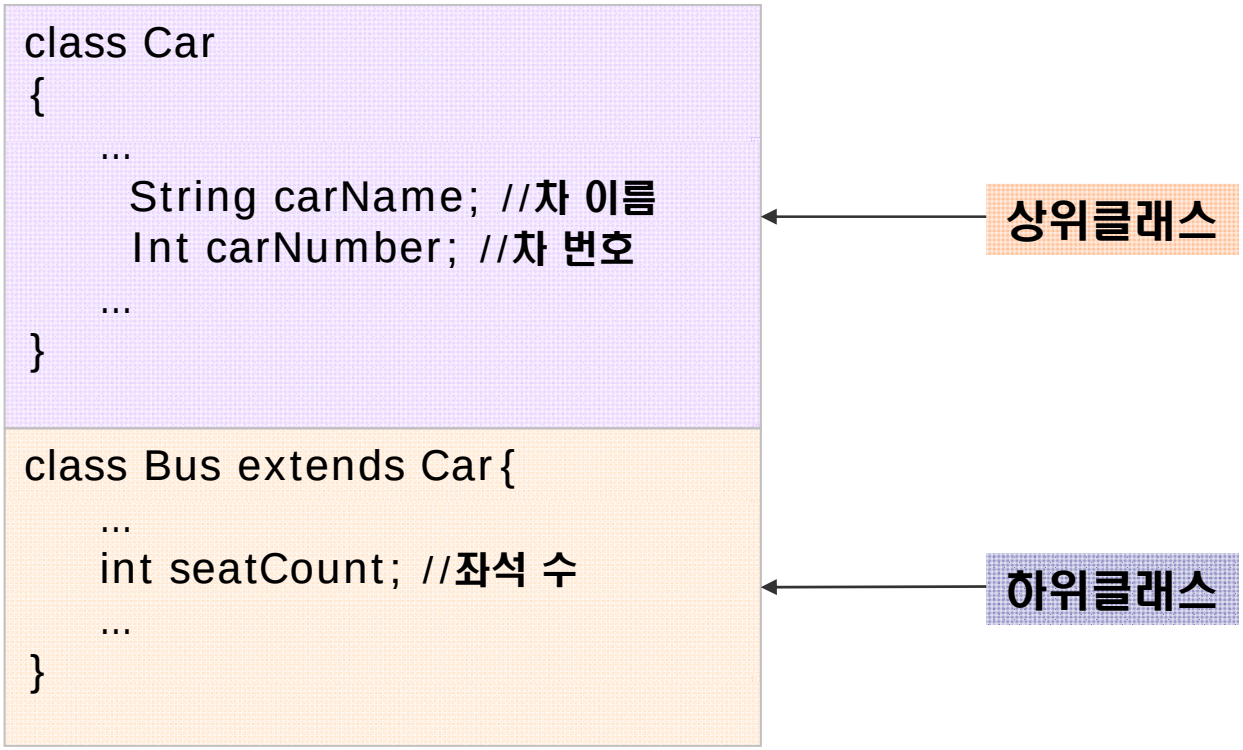


상위 클래스(Super-class)
- 바퀴가 굴러서 나아감
- 사람이나 짐을 실어 옮기는 기관

하위클래스(Sub-class)
- 각각의 구체적인 기능

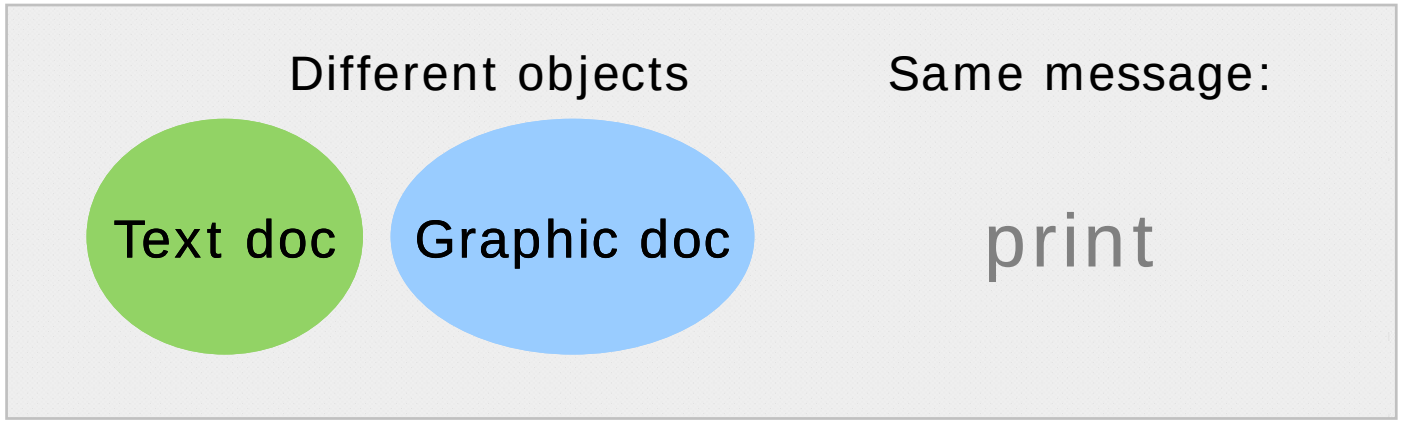
상속(Inheritance) (1/2)

❖ 상속의 예



다형성(Polymorphism)

- 의미
  - 하나의 인터페이스를 통해 서로 다른 구현을 제공하는 것



구현

구현 (1/2)

- 의미
  - 코드 작성 또는 프로그래밍이라고 함
  - 설계의 최하위 상세화 과정
  - 코드 작성, 디버깅, 통합, 개발자 테스트(단위 테스트, 통합 테스트) 작업을 포함

- 개발자의 코딩 스타일
  - 일의 효율에 영향을 끼칠 수 있음
  - 각 개발사는 코딩 스타일 지침서를 구비하여 팀원들이 지침대로 코드를 작성하도록 조율하기도 함

구현 (2/2)

- 코딩 스타일
  - 한 줄에 한 문장만 써라
  - 선언문과 실행문을 구분하라
  - 단락을 구분하라
  - 내부 블록과 피제어부는 들여써라
  - 쓸데없는 들여쓰기를 하지 마라
  - 한 줄 주석과 주석 상자를 구분하라
  - 프로그램의 앞부분에 머리 주석을 반드시 달아라
  - 함수의 역할을 접두사로 활용하라
  - 이름을 의미 있게 지어라
  - 이름은 의미를 잃지 않는 범위에서 짧게 지어라
    [좋은 코딩 나쁜 코딩 중]

연습문제

1. 설계 품질을 평가하기 위해서는 반드시 좋은 설계에 대한 기준을 세워야 한다. 좋은 설계 기준은 무엇인가?

2. 결합도(coupling)가 강한 순서대로 나열하라.

3. 한 모듈이 다른 모듈의 내부 기능 및 그 내부 자료를 참조하는 경우를 무슨 결합이라고 하는가?

4. 데이터 설계에 있어서 응집도(Cohesion)는 무슨 의미인가?

5. 효과적인 모듈화 설계 방안은 무엇인가?

6. 응집도가 강한 것부터 약한 순서로 나타내어라.

7. 모듈의 구성요소가 하나의 활동으로부터 나온 출력 자료를 그 다음 활동의 입력 자료로 사용하는 같은 모듈 내에서의 응집의 정도를 나타내는 것은 무엇인가?

8. 소프트웨어 개발 방법론에서 구현에 대해 설명하라.

팀 프로젝트

10, 11주차

이번 주 할일

- 각 팀은 설계 단계에 들어간다

- 설계 문서 평가 기준 (5점 만점)
  - 요구사항 명세서에 맞게 설계 되었는가
  - 모듈 설계가 3개 이상으로 되어 있는가
  - UML작성은 명확한가
  - 구현에 들어갈 만큼 상세화 되어 있는가

- 결과
  - 3.5점 이상이면 통과 함

다음 주 제출 문서

- 설계 문서를 제출한다