1장. 객체지향 방법론

객체지향 방법론

- 본문의 C++ 코드를 이해하기 위해
- 추상 자료형 개념과 어떻게 일치하는지 이해하기 위해
- 객체 지향 방법론의 장점은 사용과 구현을 분리하는데 있음

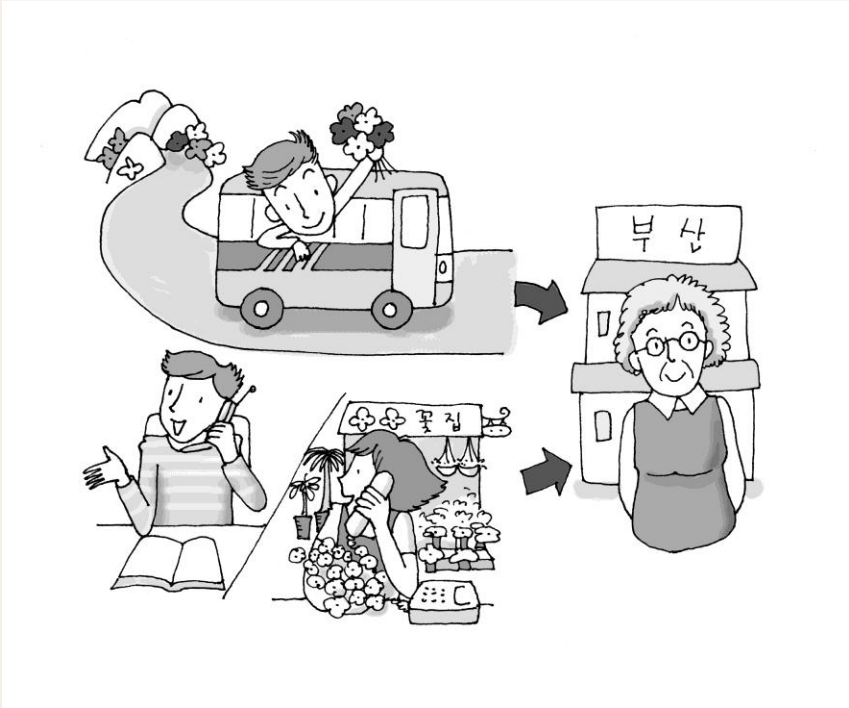
학습목표

- 프로그램 설계 차원에서 객체지향 방법론의 개념이해
- C++의 객체지향적 요소를 파악
- 인터페이스와 구현을 분리하는 이유를 이해
- 절차적 방법론과 객체지향 방법론의 차이를 이해

객체지향 방법론

- 객체지향은 프로그램 설계기법
  - 프로그램을 구현하기 위한 언어가 아님

- 문제: 부산에 사시는 할머니에게 꽃을 보낸다
  - 절차적 방법론, 절차적 언어
  - 객체지향적 방법론, 객체지향적 언어



객체지향 방법론

객체지향 용어

- 메시지, 요구사항
- 객체, 대리인, 메시지을 받는 사람
- 객체는 메시지 수신자이며 동시에 전달자
- 정보의 은닉

객체지향 방법론

- 주체지향에 대응되는 개념
- 객체 설정이 중요
  (예: 인사관리 객체, 수금관리 객체, 재고관리 객체, ...)

- 조교에게 책임을 전가

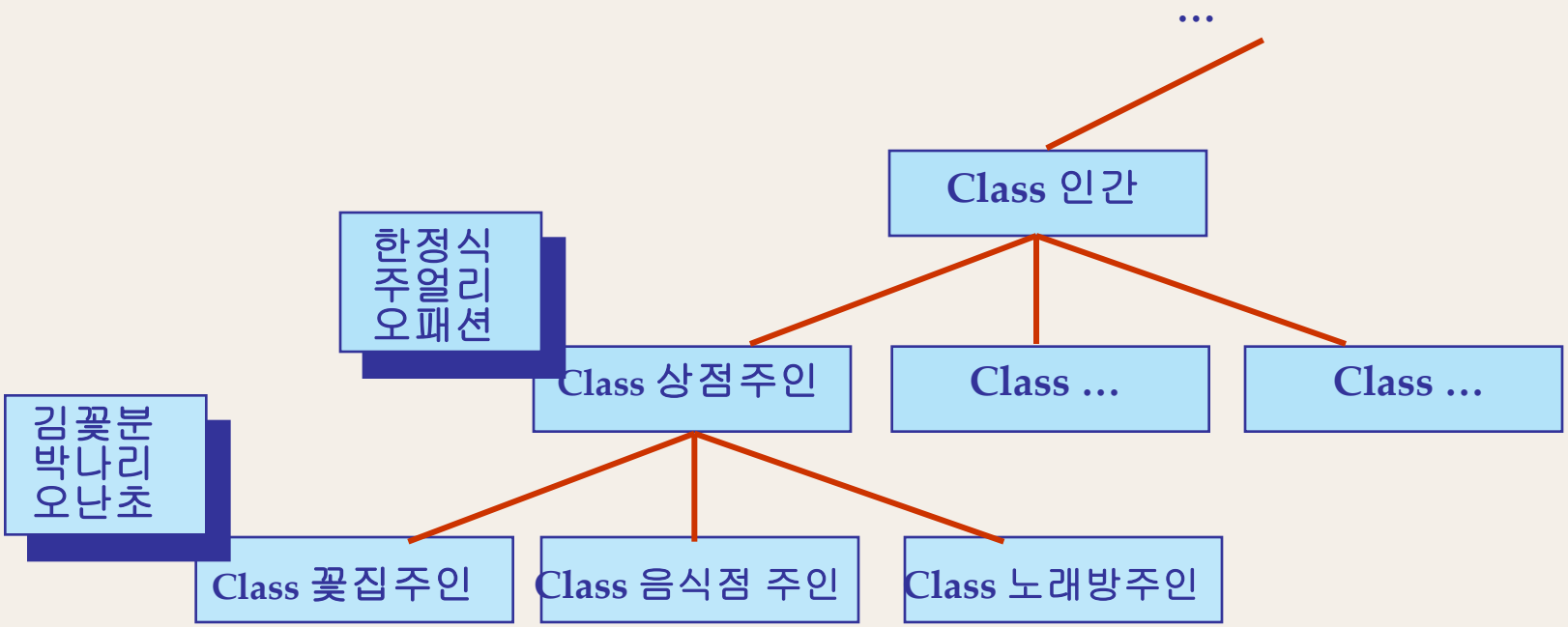
객체 클래스

- **클래스**
  - 유사한 특징을 지닌 객체를 묶어서 그룹지은 것
  - 객체 = 클래스 인스턴스
  - 같은 메시지에 같은 클래스 객체라면 동일하게 반응

- **다형성**
  - Poly-Morphism
  - 같은 메시지에 대해 클래스 별로 서로 달리 반응하게 할 수 있음

클래스 계층구조

- 상위 클래스, 하위 클래스
  - 하위 클래스를 일반화한 것이 상위 클래스
  - 상위 클래스의 특수한 경우가 하위 클래스
  - 계층구조 선언의 목적은 상속에 있음



객체지향 설계과정

- 1단계
  - 문제를 풀기위해 필요한 객체를 설정
- 2단계
  - 객체들 간의 유사속성을 파악
- 3단계
  - 유사한 속성을 지닌 객체를 모아 기본 클래스로 선언
- 4단계
  - 기본 클래스로부터 특수한 속성을 지닌 하위 클래스를 선언

C++와 객체지향

- **객체 단위로 2 개의 파일**
  - 인터페이스 파일(.h)과 구현 파일(.cpp)
  - 객체 단위의 재사용성을 높일 수 있음

- **인터페이스 파일**
  - C 용어로는 헤더파일
  - 외부 사용자를 위한 파일
  - 메시지가 정의됨
  - 구현을 몰라도 이 파일만 읽고 불러서 사용할 수 있음(정보의 은닉)
  - 제대로 된 커멘트가 중요함(이유?)

- **구현파일**
  - 내부 구현자를 위한 파일

인터페이스 파일 예

코드 1-1:

```
enum suits {diamond, clover, heart, spade} 타입 suit는 카드무늬 집합 중 하나
enum colors {red, black}                   타입 colors는 카드색 집합 중 하나


class card{
public:
    card( );                  생성자 함수
    ~card( );                 소멸자 함수
    colors Color( );          현재 카드의 색깔을 되돌려 주는 함수
    bool  IsFaceUp( );        앞면이 위인지 아래인지 되돌려주는 함수
    int   Rank( );            카드에 쓰인 숫자를 되돌려주는 함수
    void  SetRank(int x);     카드의 숫자를 x로 세팅하는 함수
    void  Draw( );            카드를 화면에 그려내는 함수
    void  Flip( );            카드를 뒤집는 멤버함수
private:
    bool  Faceup;             그림이 위로 향하고 있는지 나타내는 변수
    int   Rval;               카드 숫자를 나타내는 변수
    suits Sval;               카드 종류를 나타내는 변수
};
```

인터페이스 파일

- **인터페이스 파일**
  - 외부사용자와의 인터페이스
  - 클래스 선언 파일

- **퍼블릭 섹션**
  - 외부 사용자에게 공개된(직접 불러서 사용할 수 있는) 부분
  - 메시지 = C++ 멤버함수
  - 함수 프로토타입만 선언됨
  - 구현내용을 여기에 써서는 안됨(정보의 은닉)
  - 선언만 보고도 불러쓸 수 있도록 자세하고도 정확한 커멘트 처리

- **프라이빗 섹션**
  - 외부사용자에게 비공개된(직접 사용할 수 없는) 부분
  - 자체 클래스의 멤버함수만이 사용할 수 있음
  - 멤버 데이터(Member Data, State Variable, Instance Variable)
  - 여기에 멤버함수를 정의하면 그 함수는 자체 클래스의 멤버함수만이 사용할 수 있음

- **생성자, 소멸자 함수**

구현파일 예

코드 1-2:

```
#include "card.h"
card::card( )
{   Sval = diamind;          카드 종류는 다이아몬드
    Rval = 7;                카드 숫자는 7로
    Faceup = TRUE;           앞면을 위로
}
int card::Rank( )
{   return Rval;             현재의 카드 숫자를 되돌려 줌
}
```

구현 파일

- **클래스명, 더블 콜론(::), 메시지명**
  - 어느 클래스에 속한 함수인지를 표시함
  - 같은 이름의 메시지라 할지라도 클래스에 따라서 처리방식이 다를 수 있음

- **생성자함수**
  - 멤버 데이터 값을 초기화
  - 프라이빗 섹션의 상태변수 사용가능

C++ 소스파일 구성

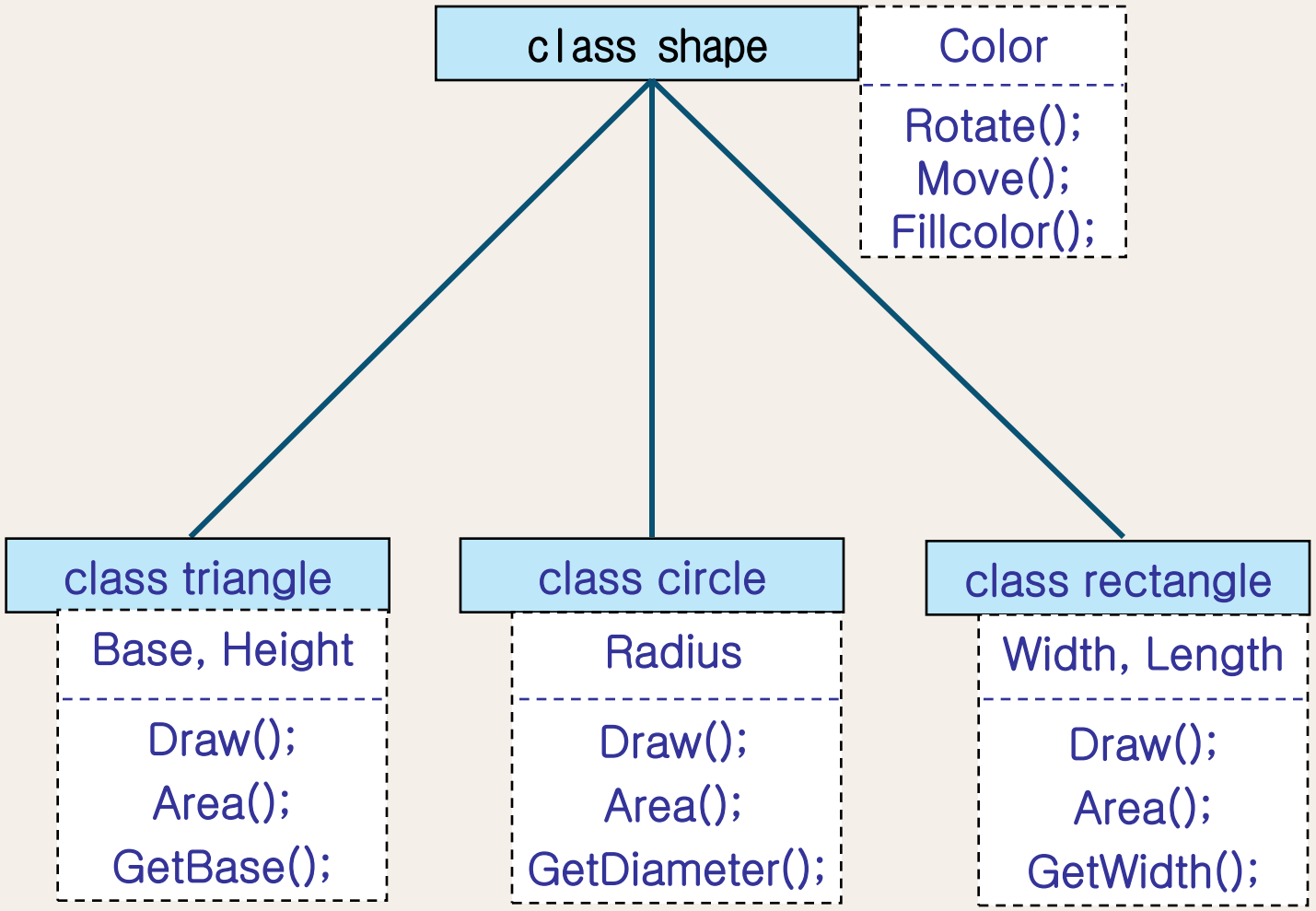
Interface File (.h)	Implementation File (.cpp or .c or .cc)
클래스 선언	클래스 구현
#define: 매크로 정의 #include: 다른 인터페이스 파일 포함 typedef: 타입선언	#include: 자체 헤더 파일 포함 데이터/ 객체 선언

메시지 전달

```
void main( )
{   card MyCard;          내 카드 객체 하나 만들기
   MyCard.Flip( );         내 카드여! 뒤집어라
   cout << MyCard.Rank( );
                              내 카드여! 현재 네 숫자가 얼마인지 화면에 찍어라.
}
```

- 메시지를 받을 객체 다음에 점을 찍고 메시지 명
  - MyCard는 객체이고 Flip( )은 그 객체가 실행할 수 있는 메시지
  - 필요하다면 괄호 안에 파라미터를 전달
  - 멤버함수 다음에는 파라미터가 없더라도 빈 괄호를 넣는 것이 상례

클래스 계층구조 예


class shape
Color
Rotate();
Move();
Fillcolor();
class triangle
Base, Height
Draw();
Area();
GetBase();
class circle
Radius
Draw();
Area();
GetDiameter();
class rectangle
Width, Length
Draw();
Area();
GetWidth();

파생 클래스 선언

```
class triangle: public shape{
public:
    triangle( );              생성자 함수
    ~triangle( );             소멸자 함수
    void  Draw( );            현재 객체를 화면에 그리는 함수
    float  Area( );           현재 객체의 면적을 계산하는 함수
    float  GetBase( );        현재 객체의 밑변 길이를 되돌려주는 함수
private:
    float Base;               밑변의 길이를 나타내는 변수
    float Height;             높이의 길이를 나타내는 변수
};
```

- class triangle: public shape
  - 클래스 triangle이 클래스 shape의 하위 클래스임을 밝힘.

멤버함수의 연결

- triangle T; T.Rotate( );
  - T.Rotate( );에 의해 일단 클래스 triangle에 해당 함수가 있는지를 검색
  - 있으면 그 클래스의 함수가 실행
  - 없으면 상위 클래스인 클래스 shape에 해당 함수가 있는지를 검색
  - 계속적으로 상위 클래스로 올라가면서 가장 먼저 정의된 것을 상속받음
  - 객체와 메시지의 연결은 실행 시에 일어남(동적 연결:Dynamic Binding)

- 하위 클래스에서 상위 클래스와 동일한 함수를 정의
  - 상위 클래스의 함수를 덮어 씌움
  - 일반적인 상위 클래스 함수를, 하위 클래스가 자체 특성에 맞게 특화(Specialization)시킬 때 사용되는 기법

- triangle T; circle C; rectangle R; T.Draw( ); C.Draw( ); R.Draw( );
  - Draw라는 명령은 동일
  - 삼각형, 원, 사각형 등 서로 다른 그림이 그려짐
  - Draw( )라는 동일 메시지를 클래스 별로 다른 방법으로 구현.(다형성)

프로텍티드 섹션

- Protected Section
  - 하위 클래스가 상위 클래스의 상태변수나 메시지를 사용해야 할 때가 있다.
  - 이를 가능하게 하려면 상위 클래스에서 이들을 protected 섹션 내부에 정의하면 된다.
  - protected 섹션에 의해서 다른 클래스 객체와 자신의 하위 객체의 접근성을 차별화한다.
  - 즉, 다른 객체는 접근할 수 없지만 같은 가족인 하위 클래스 객체는 접근을 가능하게 한 것이다.

연산자 오버로딩

- 동일한 연산자에 2개 이상의 의미를 부여
- 어떤 의미의 함수가 불려오는지는 호출 형태에 의해 결정
  - Int x, y; x = y; cardClass c1, c2; c1 = c2;
  - 정수 할당인 이퀄 연산자와 카드 할당인 이퀄 연산자는 달라야 함
  - 객체 c1, c2의 상태변수는 단순한 하나의 변수가 아닐 수 있음. 객체의 복사는 상태변수 모두를 복사해야 함. 이를 위해 이퀄 연산자에 다른 의미를 부여
  - 연산자 다음에 card 클래스의 객체 C가 나오면 오버로딩 된 이퀄 연산자가 실행됨.

card.h

```
void operator = (card C);
```

이퀄 연산자의 우변 c2에 해당하는 것이 C

card.cpp

```
void card::operator = (card C)
{ FaceUp = C.FaceUp;          c1.FaceUp = c2.FaceUp
  Rval = C.Rval;              c1.Rval = c2.Rval
  Sval = C.Sval;              c1.Sval = c2.Sval
}
```

객체지향적 언어, 절차적 언어

객체지향 언어, 절차적 언어

- 객체지향 언어는 객체를 강조
  - 카드 객체에게 뒤집는 작업을 시킬 것인가
- 절차적 언어는 작업을 강조
  - 뒤집는 함수에게 내 카드를 전달할 것인가

절차적 언어

- 자주 사용되는 작업을 함수로 정의. 반복 호출에 의해 재 사용성을 높임.
- 객체 개념이 없음
- 최소치를 구하는 함수, 평균값을 구하는 함수

객체지향적 언어

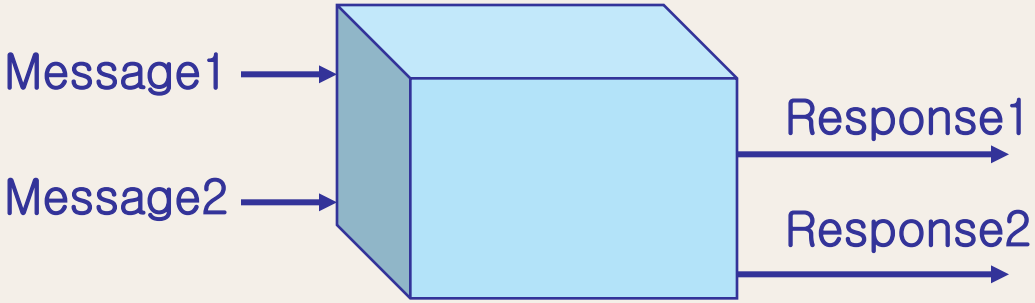
- 객체를 우선적으로 설정
- 계산기 객체를 일단 선언하고 그 클래스 객체가 실행할 수 있는 함수를 선언

구조면

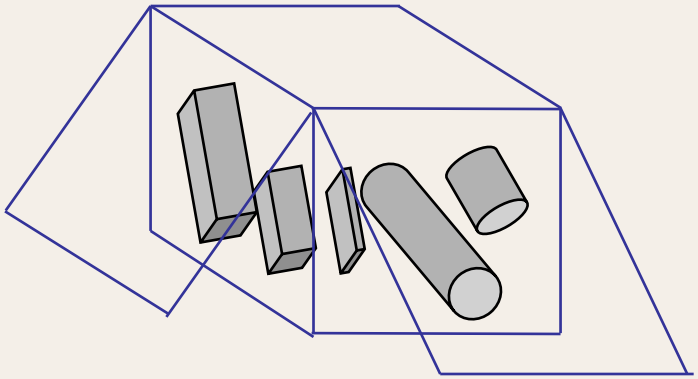
- 객체지향: 객체 울타리 안에 변수와 함수를 묶어버림
- 절차적 설계: 어느 변수가 어느 함수에 사용되는지 분간하기 힘

인터페이스와 구현의 분리

- 객체지향 설계의 최대 장점
  - 인터페이스와 구현의 분리



인터페이스 관점: Object=Black Box



구현 관점: Object=Visible