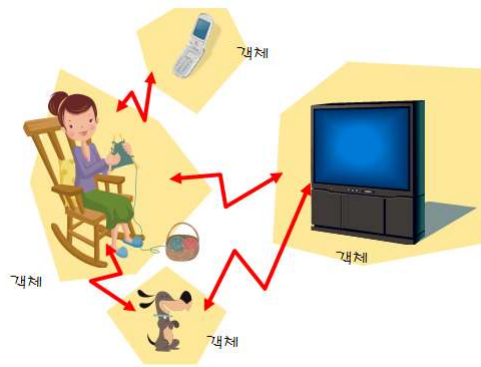


제5장 클래스의 기초



이번 장에서 학습할 내용

- 클래스와 객체
- 객체의 일생
- 메소드 → 멤버함수
- 필드 → 멤버변수
- UML

직접
클래스를
작성해
봅시다.



구조체 vs. 클래스

```
// 구조체(데이터 형), C 에서 사용
struct account {
    char owner[30]; // 통장 소유자 이름
    int money;
};

void main() {
    struct account me;    // 변수
    me.money = 12.5;
    cout << me.money;
}
```

```
// class (데이터 형) , C++ 에서 사용
class account { // 설계도
public:
    char owner[30]; // 통장 소유자 이름
    int money; // 잔액
    void Input(float m ){ // 입금하다
        money += m;
    }
};

void main( ){
    account me;    // 객체
    me.money = 100;
    me.Input(500);
    cout << me.money;
}
```

클래스의 구성 ...



- 참고)
 - type → 클래스
 - 변수 → 객체
- 클래스(class)는 객체의 설계도라 할 수 있다. → c 의 type 을 정의한 것.
 - int a; // int 저장하는 변수 생성
 - Car b; // Car 저장하는 객체 생성
- 클래스는 멤버 변수와 멤버 함수로 이루어진다.
- 멤버 변수는 객체의 속성을 나타낸다.
 - 고양이 → 나이, 이름, ...
- 멤버 함수는 객체의 동작을 나타낸다.
 - 고양이 → 먹는다. 잔다. 걷는다....



추상화

- 추상화는 많은 특징 중에서 문제 해결에 필요한 것만을 남기고 나머지는 전부 삭제하는 과정이다.
 - 자동차 속성은 많이 있지만 그 중에서 필요한 **speed, gear, color** 만 사용
 - 자동차 동작은 많이 있지만 그 중에서 필요한 **speedUp(), speedDown()** 만 사용

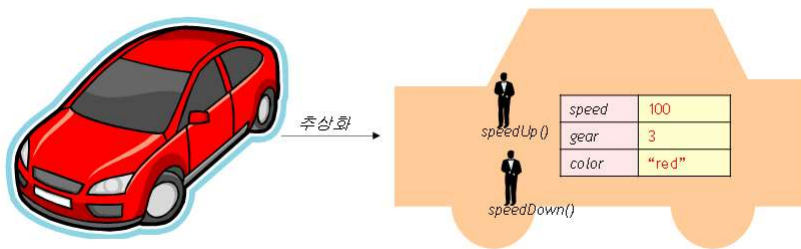


그림 4.2 추상화



클래스 정의의 예(type 을 만드는 것)



```
class Car {           // 클래스 이름은 개발자가 문제에 맞게 임의로 지정

public:               // 누구나 사용 가능, 전역
    // 멤버 변수 선언 멤버 변수 정의!
    int speed; // 속도
    int gear;  // 기어
    string color; // 색상 → 문자열로 지정 "red", "green" 등

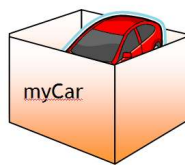
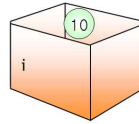
    // 멤버 함수 선언 멤버 함수 정의!
    void speedUp() { // 속도 증가 멤버 함수
        speed += 10;           // 멤버변수 값 변경하는 역할
    }

    void speedDown() { // 속도 감소 멤버 함수
        speed -= 10;
    }
}; // ← ";" 꼭 사용
```

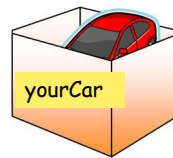


객체

- int type의 변수를 선언하는 경우
int i;
- 클래스도 type으로 생각하면 된다.
 - Car type의 변수를 선언하면 객체가 생성된다.
Car myCar, yourCar; // Car 클래스의 속성, 함수 포함하는 변수



myCar
speed, gear, color
speedUp()
speedDown()

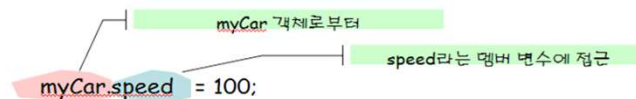


yourCar
speed, gear, color
speedUp()
speedDown()



객체의 사용

- 객체를 이용하여 멤버(멤버변수, 멤버함수)에 접근할 수 있다.



- 멤버 변수/함수에 접근하기 위해서는 도트(.) 연산자를 사용한다.
myCar.speed = 100;
myCar.speedUp();
myCar.speedDown();

- 참고 :

- Car myCar;
- 클래스이름.멤버변수/함수” 는 불가 → Car.speed = 100; // 불가
- 객체이름.멤버변수/함수” 만 가능 → myCar.spped = 100; // 가능



객체의 사용

```
Car(클래스)
int speed, gear;
string color;
speedUp( ){ speed += 10; }
speedDown( ){ speed -= 10; }
```

```
Car myCar, yourCar;
```

```
myCar.speed = 100;
```

```
myCar.speedUp();
```

```
myCar.speedDown();
```

```
myCar(객체)
speed = 100 → 110 → 100
gear, color
speedUp( ){ speed += 10; }
speedDown( ){ speed -= 10; }
```

```
yourCar.speed = 200;
```

```
yourCar.speedUp();
```

```
yourCar.speedUp();
```

```
yourCar (객체)
speed = 200 → 210 → 220
gear, color
speedUp( ){ speed += 10; }
speedDown( ){ speed -= 10; }
```



예제

```
#include <iostream>
#include <string>
using namespace std;

class Car {
public:
    // 멤버변수선언
    int speed; // 속도
    int gear; // 기어
    string color; // 색상

    // 멤버함수선언
    void speedUp(){ // 속도증가멤버함수
        speed += 10;
    }

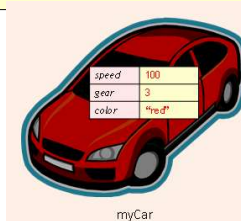
    void speedDown(){ // 속도감소멤버함수
        speed -= 10;
    }
};
```

```
int main(){
    Car myCar;

    myCar.speed = 100;
    myCar.gear = 3;
    myCar.color = "red";

    myCar.speedUp();
    myCar.speedDown();

    return 0;
}
```



여러 개의 객체 생성



```
#include <iostream>
#include <string>
using namespace std;

class Car {
public:
    int speed; // 속도
    int gear; // 기어
    string color; // 색상

    void speedUp() { // 속도증가멤버함수
        speed += 10;
    }
    void speedDown() { // 속도감소멤버함수
        speed -= 10;
    }
    void show() { // 상태출력멤버함수
        cout << "===== " << endl;
        cout << "속도: " << speed << endl;
        cout << "기어: " << gear << endl;
        cout << "색상: " << color << endl;
        cout << "===== " << endl;
    }
};
```



예제



```
int main()
{
    Car myCar, yourCar;

    myCar.speed = 100;
    myCar.gear = 3;
    myCar.color = "red";

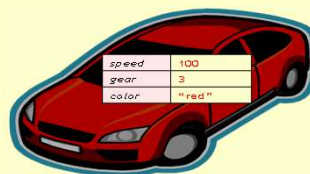
    yourCar.speed = 10;
    yourCar.gear = 1;
    yourCar.color = "blue";

    myCar.speedUp();
    yourCar.speedUp();

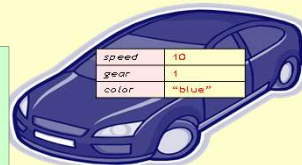
    myCar.show();
    yourCar.show();

    return 0;
}
```

```
=====
속도: 110
기어: 3
색상: red
=====
속도: 20
기어: 1
색상: blue
=====
```



myCar



yourCar



객체의 동적 생성

- C 에서 변수 동적 생성

```
int *x = new int;           // 변수 동적 생성
int *y = new int[10];       // 변수 동적 생성

*x = 100;
Y[3] = 10;
...
delete x;                   // 동적 변수 삭제
delete[] y;
```

	주소	값
x	1000	1001
	1001	100
y	1002	1003
	1003	
	1004	
	1005	
	1006	10
	...	



객체의 동적 생성

- 객체도 new와 delete를 사용하여 동적으로 생성할 수 있다.

```
Car *dynCar = new Car;      // 동적 객체 생성
dynCar->speed = 100;        // 동적 객체 사용,
dynCar->speedUp();           // 포인터 가리키는 객체의 멤버 지칭시 ">" 사용
...
delete dynCar;              // 동적 객체 삭제
```

- 객체가 포인터 아닌 경우 멤버 접근

```
Car dynCar;
dynCar.speed = 100;
```

- 객체가 포인터인 경우 멤버 접근

```
Car *dynCar= new Car;
dynCar->speed = 100;
```

	주소	값
dynCar	1000	1002
	1001	
	1002	speed = 100 color, gear -Up(), ...
	1003	



예제

```
int main(){
    int w;
    int *x = new int;
    int *z = new int[5];

    w = 10;
    z[2] = 200;    // *(z+2) = 200; 동일
                  // *(1004+2)
    *x = 100;      // 포인터 인 경우

    delete x;
    delete []z;

    return 0;
}
```

	주소	값
w	1000	10
x	1001	1002
	1002	100
z	1003	1004
z[0]	1004	
z[1]	1005	
z[2]	1006	200
...	1007	
	...	



예제

```
class Car {
public:
    int speed; // 속도

    void speedUp() { // 속도증가멤버함수
        speed += 10;
    }

    void print() {
        cout << speed << endl;
    }
};
```

참고) 객체의 멤버 지칭시
포인터 객체 아니면 "." 사용
포인터 객체 이면 ">" 사용

```
int main(){
    Car w;
    Car *x = new Car;
    Car *z = new Car[4];

    w.speed = 10;

    (z+2)->speed = 200;
    z[2].speedUp(); // z[2].speed = 210
    z[2].print();

    x->speed = 100;
    x->speedUp();    x->print();

    delete x;
    delete []z;

    return 0;
}
```



중간 점검 문제

- 강아지를 나타내는 클래스(CDog)를 작성하여 보라.
 - 강아지의 이름, 종 등을 멤버 변수로 지정하고 (문자열 **string** 사용)
 - 짓기, **Show** 등의 멤버 함수를 정의하라.
 - 짓기는 “멍멍” 을 출력 → 함수이름 **A()**
 - Show()** 는 이름, 종을 출력
 - main** 에서 개 두 마리(**x, y**) 만들고 적당한 속성값 지정 후 각 객체의 **A**, **Show** 함수 호출하라.
- main** 에서 **yourDog** 이라는 객체를 다음과 같이 생성하고 속성을 지정한 후 각 객체의 함수들을 실행하라.
 - CDog *yourDog;**
 - yourDog = new CDog;**



#include <string> 필요

```
class CDog {
public:
```

멤버 변수

멤버 함수

```
};
```

```
int main(){
```

```
    CDog x, y;
```

객체 **x, y**

```
    CDog *yourDog;
```

```
    yourDog = new CDog;
```

객체 **yourDog**

```
    return 0;
```

```
}
```

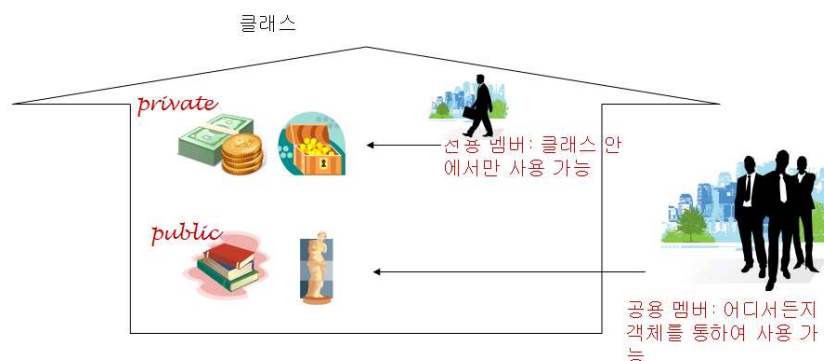


접근 제어..

- **public vs. private**
- 세상의 모든 물체들은 내부로의(내부 부품) 접근을 대부분 불허
- TV 경우
 - TV 내부에는 많은 부품들이 존재, 그러나 외부에서 부품들 직접 접근 못함(예를 들면, 채널 변경시 내부 부품 직접 조작 못함) → 내부 부품들은 **private**
 - 외부에서 접근할 수 있는 것 → 전원버튼, 소리버튼, 채널버튼 → **public**
- 클래스는 세상의 모든 물체들을 모델링 위한 것 → **public** 을 최소화
- 클래스의 **public, private**
 - **public** → 클래스 안의 멤버 변수, 멤버함수를 누구나 사용 가능
 - **private** → 클래스 안의 멤버 변수, 멤버함수는 그 클래스 안에 있는 멤버함수만 사용 가능, **같은 클래스의 객체들(멤버함수)도 사용 가능**
 - 사용/접근 → 변수에 값 저장, 변수값 읽기, 함수 호출



접근 제어..



- **public** → 멤버 변수, 멤버함수를 누구나 사용 가능
- **private** → 멤버 변수, 멤버함수는 그 클래스 안에 있는 멤버함수만 사용 가능
 - 그 클래스 안의 멤버함수만 접근 가능 → 같은 클래스의 다른 객체도 동일한 멤버함수 소유 → 접근 가능
- 사용 → 변수에 값 저장, 변수값 읽기, 함수 호출



private와 public

```
class Car {
    // 멤버 변수 선언
    int speed; // 속도
    int gear; // 기어
    string color; // 색상
}

int main()
{
    Car myCar;
    myCar.speed = 100; // 오류!
}
```

아무것도 지정하지 않으면 디폴트로 private
일반적으로 변수는 private 사용 해야함
→ (정보은닉)

```
class Car {
public:
    int speed; // 속도
    int gear; // 기어
    string color; // 색상
}

int main()
{
    Car myCar;
    myCar.speed = 100; // OK!
}
```

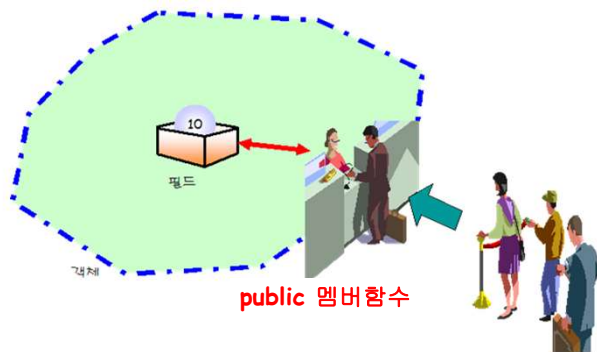
다른 지정자가 나오기 전까지
public



private와 public

```
class Car {
public:
    int speed; // 공용 멤버
private:
    int gear; // 전용 멤버
    string color; // 전용 멤버
}
```

- 변수 대부분 → private
- 함수는 외부 연결 필요한 것만 public
→ cf) TV 스위치



예제

```
#include <iostream>
#include <string>
using namespace std;
class Employee {
    string name;      // private 로 선언
    int salary, age;
    int getSalary() { return salary; } // 직원의 월급을 반환
public:
    int getAge() { return age; } // 직원의 나이를 반환
    string getName() { return name; } // 직원의 이름을 반환
};

int main() {
    Employee e;
    e.salary = 300; // 오류! private 변수
    e.age = 26;     // 오류! private 변수

    int sa = e.getSalary(); // 오류! private 멤버함수
    string s = e.getName(); // OK! → but 값 없음.
    int a = e.getAge();     // OK
}
```

멤버변수 값을
지정하지 못함



멤버 변수, 멤버함수 순서

- 일반적으로 멤버변수, 멤버함수 순으로 작성

```
class Date {
public:
    void printDate() {
        cout << year << "." << month << "." << day << endl;
    }

    int getDay() {
        return day;
    }

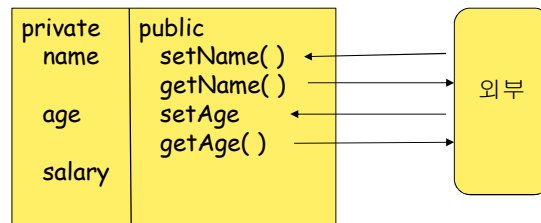
    // 멤버 변수 선언
    int year;
    string month;
    int day;
};
```

선언 위치와는 상관없이 어디서나
사용이 가능하다.



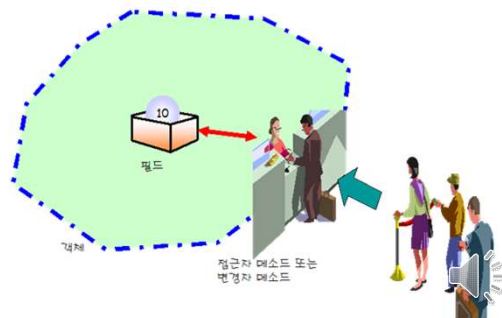
private 변수 access

- 멤버변수가 **private**,
- **but** 외부에서 접근할 필요가 있는 경우 다음 함수들이 **public** 으로 필요 ← **public** 이므로 외부에서 호출 가능
 - 변수값을 외부에서 전달된 값으로 변경하는 함수
 - 변수값을 읽어서 외부에 알려주는 함수

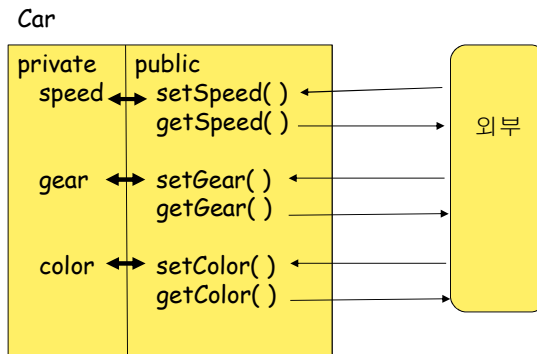


접근자와 설정자 - public 함수

- **private** 멤버변수에 외부에서 지정된 값을 저장하거나, 외부에서 변수 값을 읽어야 할 때 사용
 - 이런 함수는 **public** 으로 선언되어야 외부에서 호출 가능
- 접근자(**accessor**): 멤버 변수의 값을 반환, 외부에 전달
(예) `getAge()`, `getName()`, `getBalance()` cf) `Balance()`(잔고)
- 설정자(**mutator**): 멤버 변수의 값을 설정, 외부에서 값을 받아 저장
(예) `setAge()`, `setName()`, `setBalance()`;



다음 쪽 예제



예제

```
class Car {
private:
    int speed;      //속도      // 멤버 변수 선언
    int gear;       //기어
    string color;   //색상
    ...
public:
    // 접근자 선언 → private 인 멤버변수값을 외부에 전달하는 역할
    int getSpeed() {
        return speed;
    }
    // 설정자 선언 → private 인 멤버변수에 외부에서 전달한 값을 저장하는 역할
    void setSpeed(int s) {
        speed = s;
    }
    int getGear() {      return gear;      }
    void setGear(int g) { gear = g;        }
    string getColor() {  return color;     }
    void setColor(string c) { color = c;    }
};
```



예제

```
int main(){
    Car x;
    x.speed = 100;           // error
    x.setSpeed(100);

    x.color = "red";        // error
    x.setColor("red");

    cout << x.speed;         // error
    cout << x.getSpeed();

    cout << x.color;         // error
    cout << x.getColor();

    return 0;
}
```

```
class Car {
private:
    int speed, gear; //속도
    string color;    //색상
public:
    void setGear(int g) { gear = g; }
    void setSpeed(int s) { speed = s; }
    void setColor(string c) { color = c; }

    int getSpeed() { return speed; }
    int getGear() { return gear; }
    string getColor() { return color; }
};
```



접근자와 설정자의 장점

- 정보은닉 : TV(내부 부품, 외부 버튼)
- 설정자의 매개 변수를 통하여 잘못된 값이 넘어오는 경우, 이를 사전에 차단할 수 있다.

```
void setSpeed(int s)
{
    if( s < 0 )
        speed = 0;
    else
        speed = s;
}
```



중간 점검 문제

2. 강아지(속성 : 이름, 나이 → **private**)를 클래스로 모델링하고 **main** 에서 강아지 **x, y** 를 만들고 **x, y** 에 속성(이름, 나이)을 저장하고 이 값들을 출력하는 프로그램

```
class CDog {
private:
    string name;
    int age;

public:
    ;
};
```

```
int main(){
    CDog x, y;

    x.setName("짱");           x.setAge(2);
    y.setName("멍멍이");       y.setAge(5);

    cout << "x : " << x.getName() << " " << x.getAge() << endl;
    cout << "y : " << y.getName() << " " << y.getAge() << endl;

    return 0;
}
```



중간 점검 문제

2. 강아지(속성 : 이름, 나이 → **private**)를 클래스로 모델링하고 **main** 에서 강아지 **x, y** 를 만들고 **x, y** 에 속성(이름, 나이)을 저장하고 이를 출력(**멤버함수 show() 이용 멤버변수들 출력**)

```
class CDog {
private:

public:

    ;
};
```

```
int main(){
    CDog x, y;

    x.setName("짱");           x.setAge(2);
    y.setName("멍멍이");       y.setAge(5);

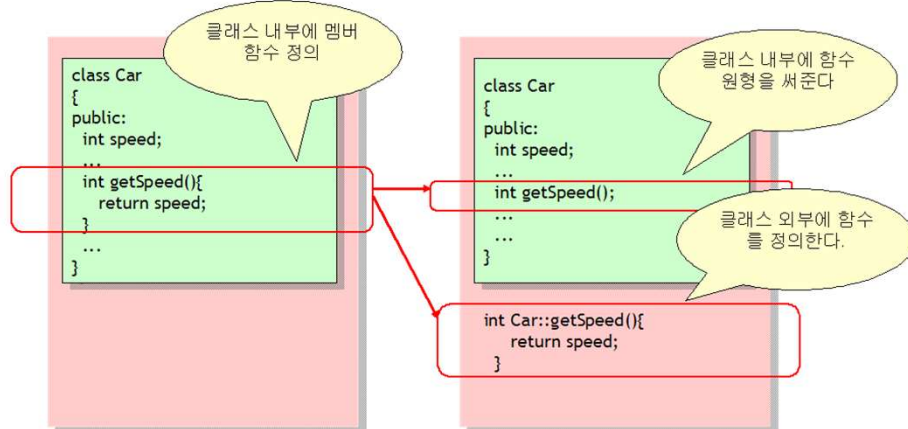
    cout << "x : " << x.show() << endl;
    cout << "y : " << y.show() << endl;

    return 0;
}
```



멤버 함수의 외부 정의 (멤버변수는 항상 내부에 정의)

- 멤버함수를 클래스 안에 작성하는 경우 멤버함수들이 길면 클래스 파악이 어렵기 때문에 클래스 밖에 작성하는 것이 일반적(외부 선언시 클래스 이름::함수 이름)



예제

멤버함수를 클래스 외부에 선언시 "클래스 이름::함수 이름"
→ 클래스가 여러 개인 경우, 멤버함수가 어떤 클래스에 속하는지 파악 가능



```
#include <iostream>
using namespace std;

class Car {
public:
    int getSpeed();
    void setSpeed(int s);
    void honk();
private:
    int speed; //속도
};

int Car::getSpeed()
{
    return speed;
}

void Car::setSpeed(int s)
{
    speed = s;
}
```

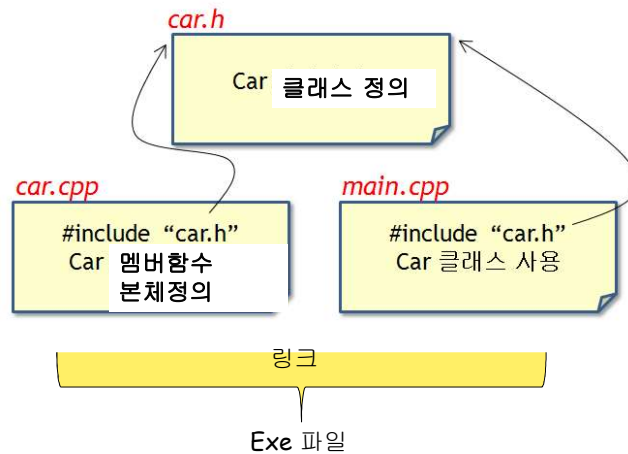
```
void Car::honk() {
    cout << "뽕뽕!" << endl;
}

int main()
{
    Car myCar;
    myCar.setSpeed(80);
    myCar.honk();
    cout << "현재 속도는" << myCar.getSpeed() << endl;
    return 0;
}
```

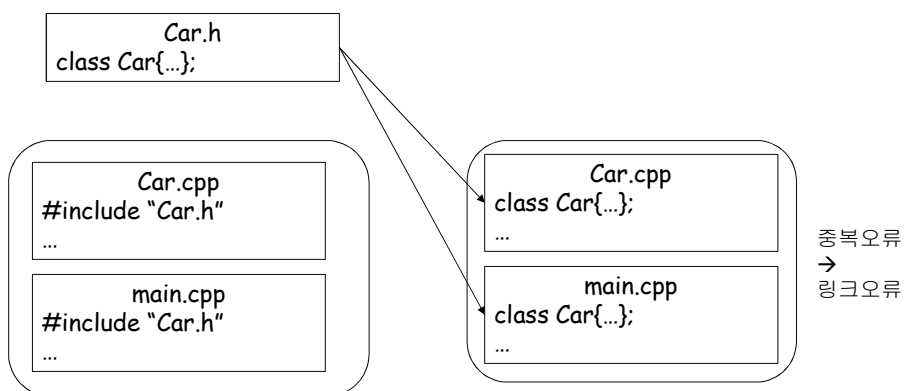
뽕뽕!
현재 속도는 80
계속하려면 아무 키나 누르십시오 ...

클래스 선언과 구현의 분리

- 클래스의 선언과 구현을 분리하는 것이 일반적

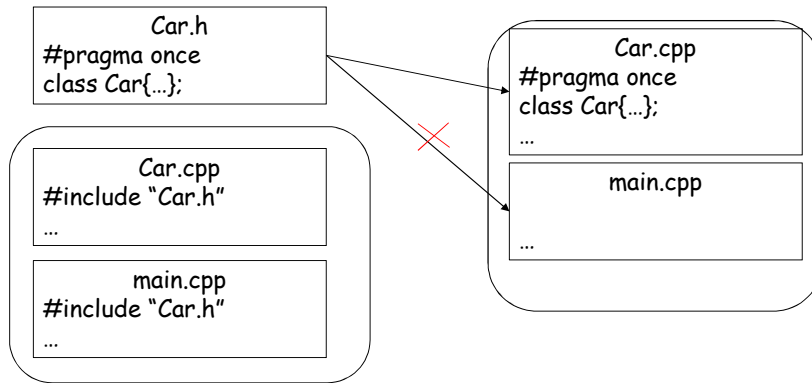


헤더파일 #pragma once



헤더파일 #pragma once

include 를 한번만 수행



예제

car.h → 클래스를 선언(정의).

```

#pragma once // include 한번만 해라
using namespace std;
class Car {
public:
    int getSpeed();
    void setSpeed(int s);
    void honk();
private:
    int speed; //속도
};
    
```

```

#include <iostream> // <...> 와 "..." 차이는 ?
#include "car.h"
// 현재 위치에 car.h를 읽어서 넣으라는 것을 의미한다.
using namespace std;
    
```

```

int main() {
    Car myCar;
    myCar.setSpeed(80);
    myCar.honk();
    cout << "현재속도는" << myCar.getSpeed() << endl;
    return 0;
}
    
```

main.cpp
클래스를 사용한다.

car.cpp → 멤버함수 본체

```

#include <iostream>
#include "car.h"
using namespace std;

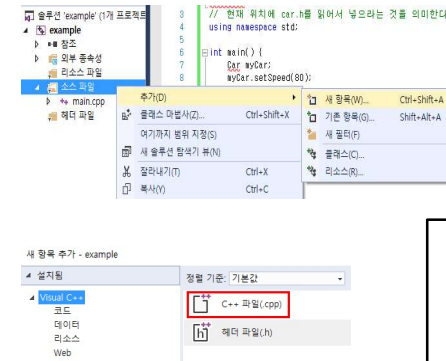
int Car::getSpeed() {
    return speed;
}
void Car::setSpeed(int s) {
    speed = s;
}
void Car::honk() {
    cout << "뽕뽕!" << endl;
}
    
```

참고) 헤더파일에서 string 사용하려면 ??

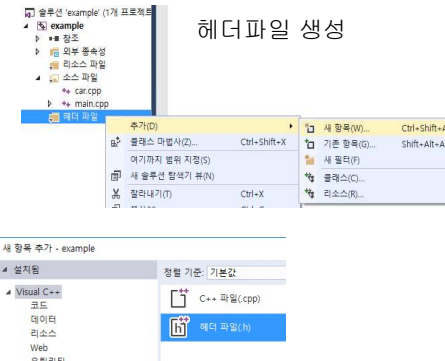
1. string 은 std 공간에 정의되어 있음
→ using namespace std; 작성
2. string.h 헤더파일 include



VS 에서 여러파일 만들기



소스파일 생성



헤더파일 생성

헤더파일 생성하면 자동으로 **#pragma once** 있음

멤버 함수의 중복 정의

- 멤버 함수도 중복 정의(오버로딩)가 가능함 → 매개변수가 달라야 함.

```
class Car {
private:
    int speed;
    int gear;
    string color;
public:
    int getSpeed();
    void setSpeed(int s);
    void setSpeed(double s);
};
```

```
int Car::getSpeed(){
    return speed;
}
void Car::setSpeed(int s) {
    speed = s;
}
void Car::setSpeed(double s) {
    speed = (int)s;
}
int main()
{
    Car myCar;
    myCar.setSpeed(80);
    myCar.setSpeed(100.0);
    cout << "차 속도: " << myCar.getSpeed() << endl;
    return 0;
}
```

UML - 생략

- UML(Unified Modeling Language): 애플리케이션을 구성하는 클래스들간의 관계를 그리기 위하여 사용

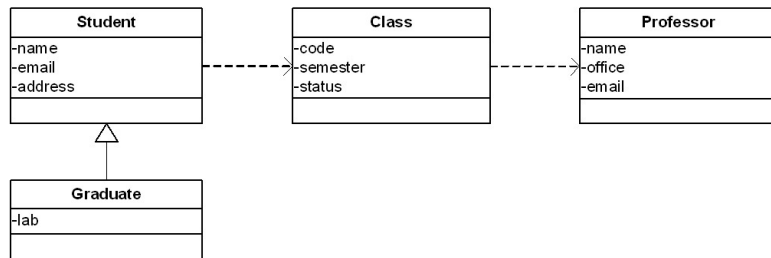


그림 8.9 UML의 예



구조체 - 생략

- 구조체(structure) --> 클래스와 유사

```
struct BankAccount { // 은행계좌
    int accountNumber; // 계좌번호
    int balance; // 잔액을 표시하는 변수
    double interest_rate; // 연이자
    double get_interrest(int days){
        return (balance*interest_rate)*((double)days/365.0);
    }
};
```

모든 멤버가 디폴트로 public이 된다.

