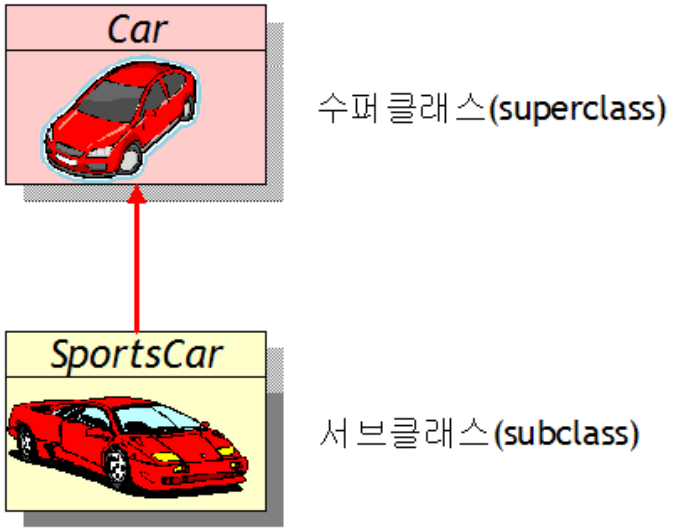
클래스의 상속
Inheritance

2023

국민대학교 소프트웨어학부

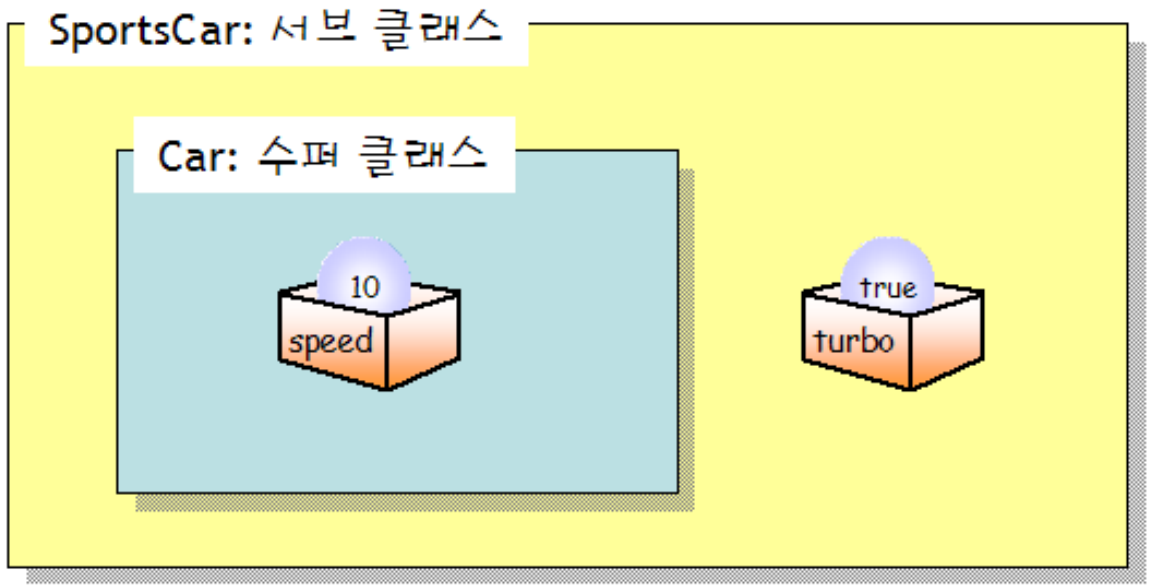
상속 : superclass 의 멤버들을 subclass 도 갖는다.

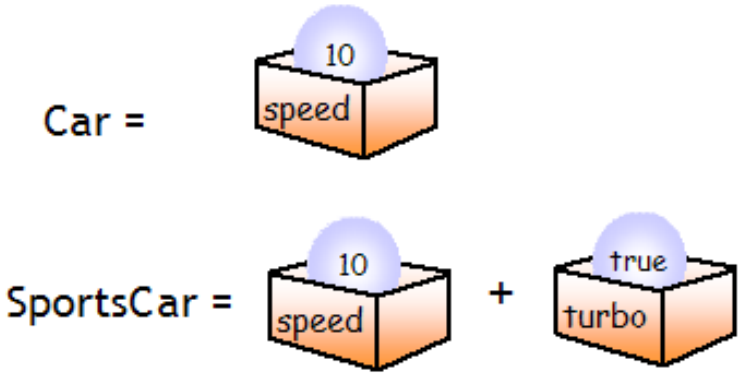
```
class Car
{
        int speed;
}
class SportsCar : public Car
{
        bool turbo;
}
```



상속한다는 의미

서브클래스는 수퍼클래스를 포함
= 서브클래스는 수퍼클래스의 확장





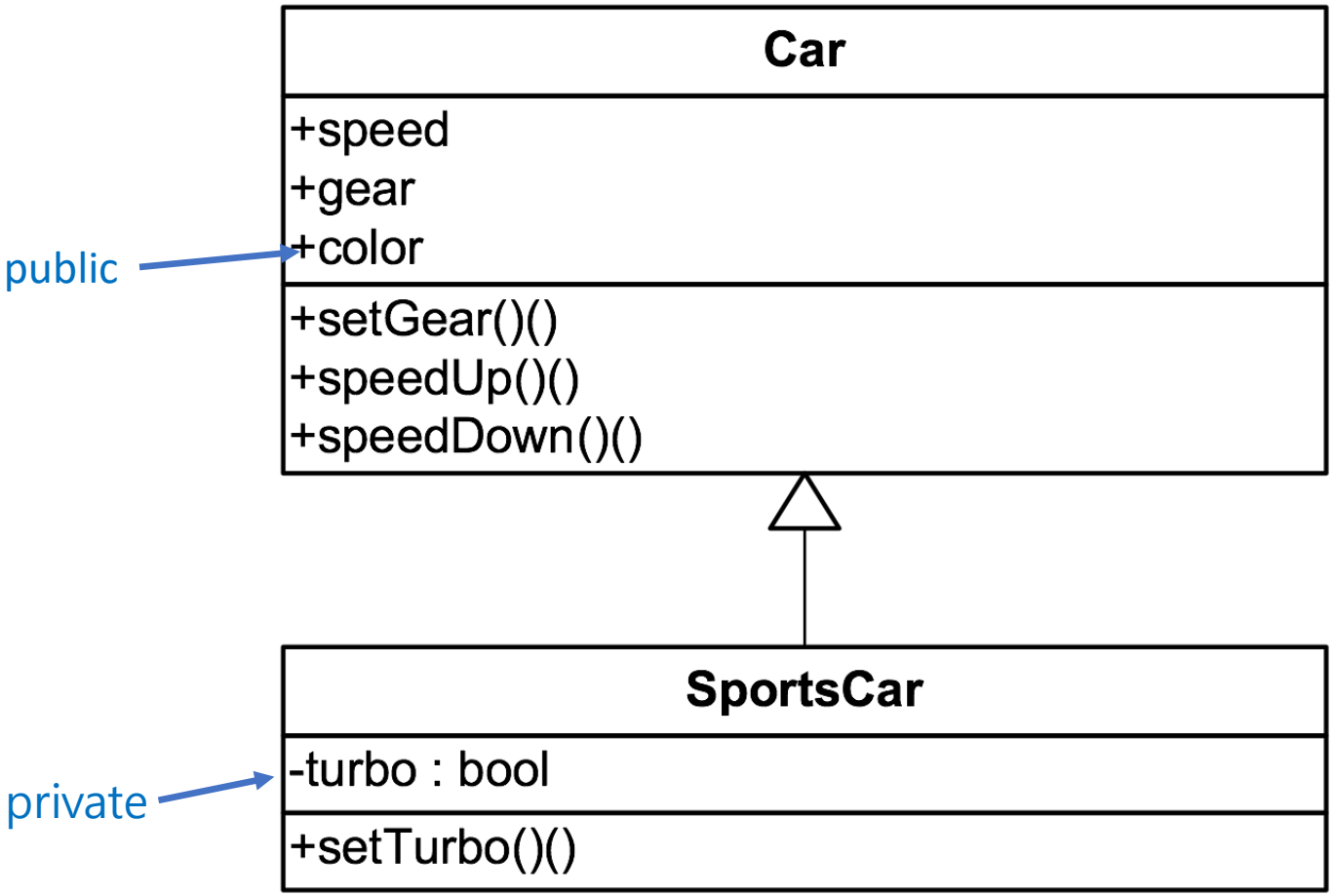
상속의 예

수퍼 클래스	서브 클래스
Animal(동물)	Lion(사자), Dog(개), Cat(고양이)
Bike(자전거)	MountainBike(산악자전거)
Vehicle(탈것)	Car(자동차), Bus(버스), Truck(트럭), Boat(보트), Motocycle(오토바이), Bicycle(자전거)
Student(학생)	GraduateStudent(대학원생), UnderGraduate(학부생)
Employee(직원)	Manager(관리자)
Shape(도형)	Rectangle(사각형), Triangle(삼각형), Circle(원)

참고

- 수퍼 클래스 == 부모 클래스(parent class) == 베이스 클래스(base class)
- 서브 클래스 == 자식 클래스(child class) == 파생된 클래스(derived class)

상속의 예제

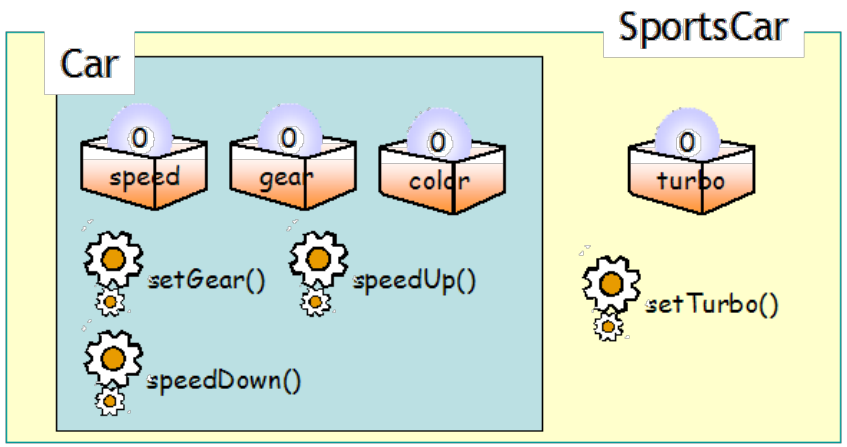

Car
+speed
+gear
+color
+setGear()()
+speedUp()()
+speedDown()()
public
SportsCar
-turbo : bool
+setTurbo()()
private

Car 클래스

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

class Car {
public:
	// 3개의 멤버 변수 선언
	int speed; // 속도
	int gear; // 주행거리
	string color; // 색상

	// 3개의 멤버 함수 선언
	void setGear(int newGear) { // 기어 설정 멤버 함수
		gear = newGear;
	}
	void speedUp(int increment) { // 속도 증가 멤버 함수
		speed += increment;
	}
	void speedDown(int decrement) { // 속도 감소 멤버 함수
		speed -= decrement;
	}
};
```

SportsCar 클래스

```
// Car 클래스를 상속받아서 다음과 같이 SportsCar 클래스를 작성하여 보자.
class SportsCar : public Car {                    // Car를 상속받는다.
      // 1개의 멤버 변수를 추가
    bool turbo;

public:
      // 1개의 멤버 함수를 추가
      void setTurbo(bool newValue) { // 터보 모드 설정 멤버 함수
                turbo = newValue;
      }
};
```

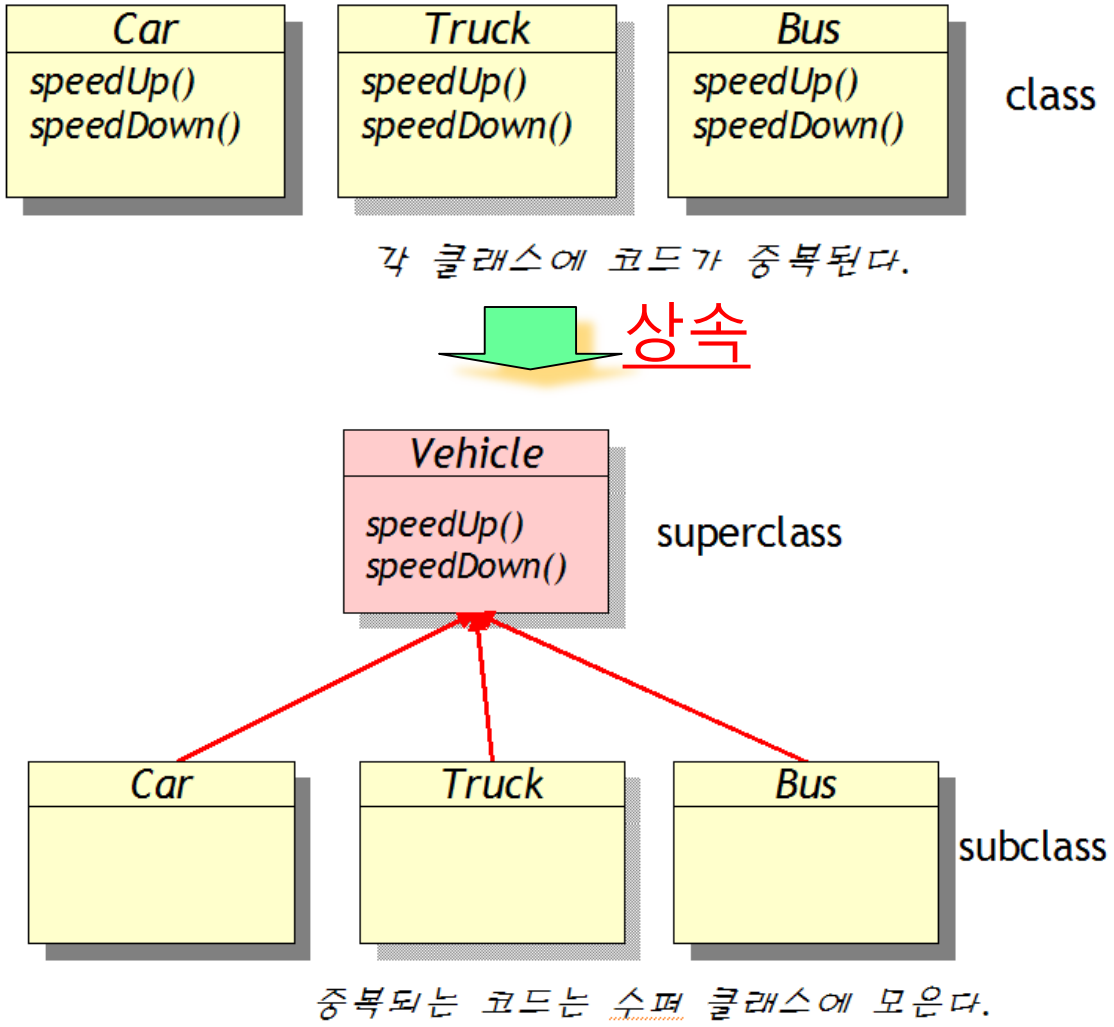


SportsCar 클래스

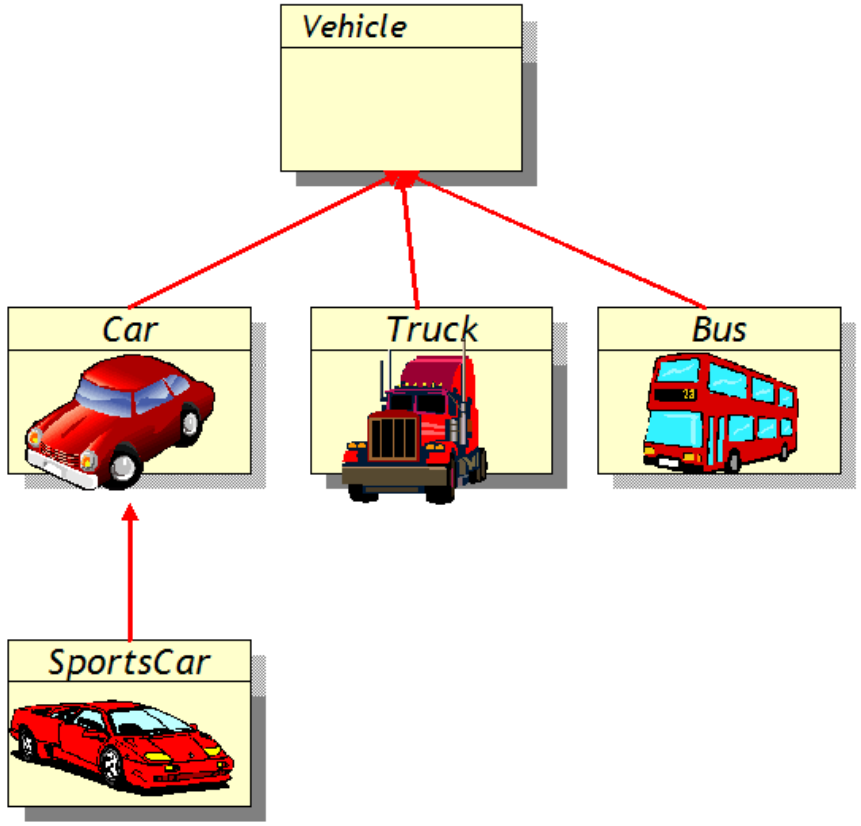
```
int main()
{
	SportsCar  c;
	c.color = "Red";		// 수퍼 클래스 멤버 변수 접근
	c.setGear(3);			// 수퍼 클래스 멤버 함수 호출
	c.speedUp(100);			// 수퍼 클래스 멤버 함수 호출
	c.speedDown(30);		// 수퍼 클래스 멤버 함수 호출
	c.setTurbo(true);		// 자체 멤버 함수 호출
	return 0;
}
```

서브클래스는 수퍼클래스의 변수와 함수를 마치 자기 것처럼 사용할 수 있다.

상속은 중복을 줄인다.


Car
speedUp()
speedDown()
Truck
speedUp()
speedDown()
Bus
speedUp()
speedDown()
class
각 클래스에 코드가 중복된다.
상속
Vehicle
speedUp()
speedDown()
superclass
Car
Truck
Bus
subclass
중복되는 코드는 수퍼 클래스에 모은다.

상속 계층도



subclass 는 superclass 의 특수화(specialization),
superclass 는 subclass 의 일반화(generalization)

상속 계층도

```
class Vehicle { ...        }
class Car : public Vehicle { ...    }
class Truck : public Vehicle { ...  }
class Bus : public Vehicle { ...    }
class SportsCar : public Car { ...  }
```

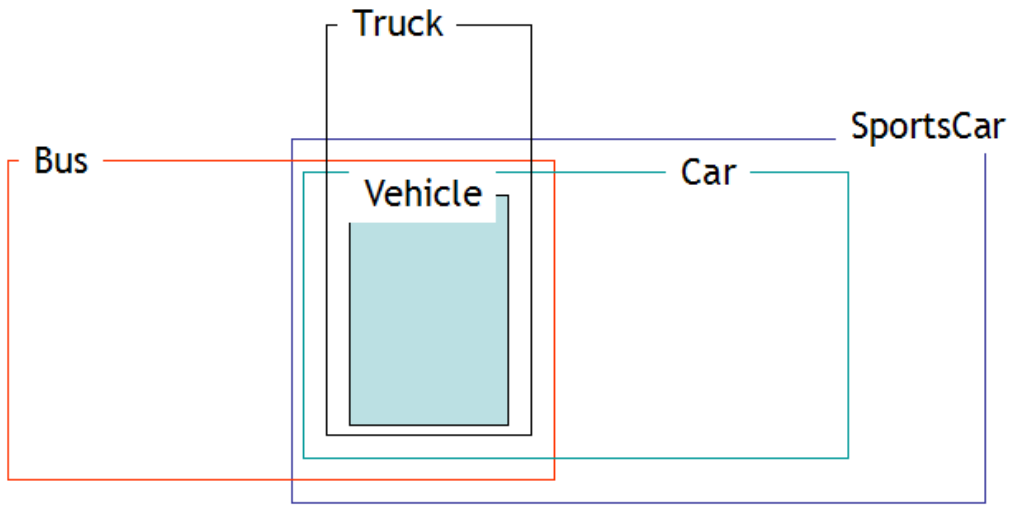


그림 13.8 클래스들의 크기

상속은 is-a 관계

- 상속에서 서브클래스와 수퍼클래스는 “~은 ~이다“와 같은 is-a 관계가 있다.
- 자동차는 탈것이다. (Car *is a* Vehicle).
- 사자, 개, 고양이는 동물이다.

```
class Chair: public Furniture{
...
};
```

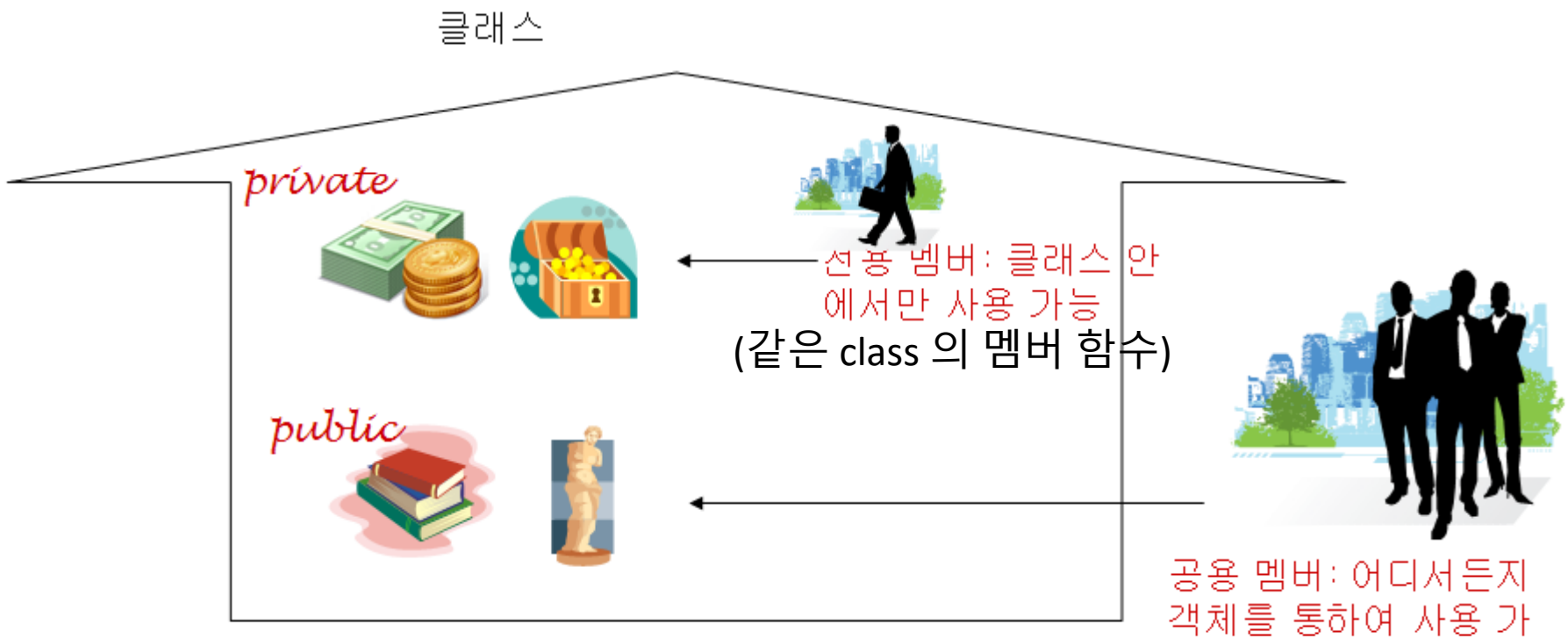
- 만약 “~은 ~을 가지고 있다“와 같은 has-a(포함) 관계가 성립되면 이 관계는 상속으로 모델링을 하면 안 된다. 예를 들어서 다음과 같다.
- 도서관은 책을 가지고 있다(Library *has a* book).
- 거실은 소파를 가지고 있다.

```
class Room {
  Chair c1, 2, c3;
};
```

상속의 장점

- 상속의 장점
  - 상속은 이미 작성된 검증된 소프트웨어를 재사용
  - 신뢰성 있는 소프트웨어를 손쉽게 개발, 유지 보수
  - 상속을 통하여 기존 클래스의 필드와 메소드를 재사용
  - 코드의 중복을 줄일 수 있다.

접근 제어자 private vs. public



선용 멤버: 클래스 안에서만 사용 가능
(같은 class 의 멤버 함수)

공용 멤버: 어디서든지 객체를 통하여 사용 가능
(외부 함수, 다른 class의 멤버 함수)

상속을 고려한 접근 제어 지정자의 확장 protected

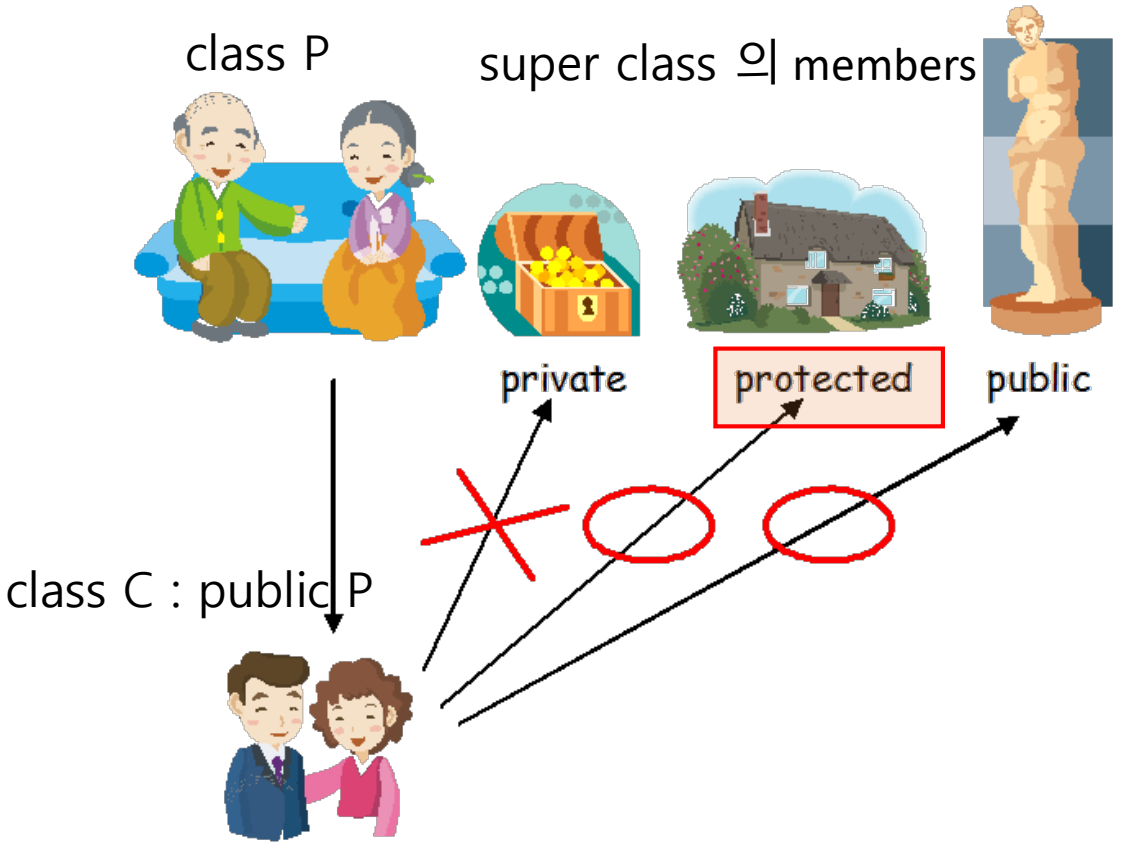
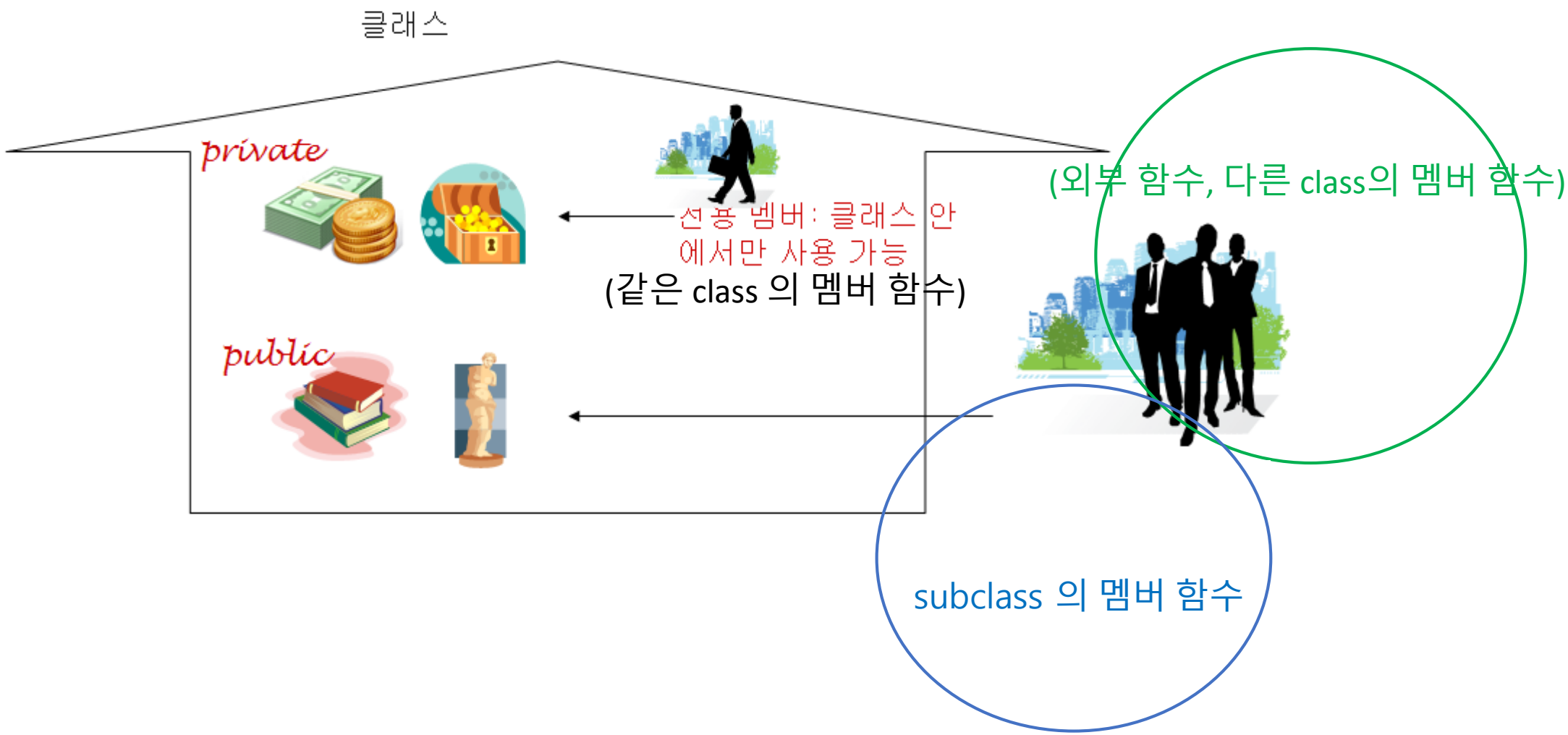


그림 13.9 상속에서의 접근 지정자

상속을 고려한 접근 제어자 private – protected - public



class A 의 멤버에 대한 접근 제어 지정자

접근 지정자	현재 클래스 A 의 멤버 함수		클래스 B 의 멤버 함수/전역 함수
private	O	×	×
protected	O	O	×
public	O	O	O

protected 는 private 과 같이 취급

집 안

```
class A {
  private : int private_v;
  protected : int protected_v;
  public: int public_v;
   void ftn1 (){
     private_v = …      // O
     protected_v = …   // O
     public_v = …       // O
   }
};
```

집 밖

```
class B{
  void ftn3(){
    A  obj1;
    obj1.private_v =… // X
    obj1.protected_v =… // X
    obj1.public_v = … // O
  }
};
```

(상속을 고려한) class A 의 멤버에 대한 접근 제어 지정자

접근 지정자	현재 클래스 A 의 멤버 함수	서브클래스 subA 의 멤버 함수	클래스 B 의 멤버 함수/전역 함수
private	o	×	×
protected	o	o	×
public	o	o	o

protected 는 상속에서 의미가 있음

```
class A {
  private : int private_v;
  protected : int protected_v;
  public: int public_v;
   void ftn1 (){
     private_v = …      // O
     protected_v = …   // O
     public_v = …       // O
   }
};
```

```
class subA : public A {
 void ftn2(){
   private_v = … // X
   protected_v = … // O
   public_v = … // O
 }
};
```

```
class B{
  void ftn3(){
    A  obj1;
    obj1.private_v =… // X
    obj1.protected_v =… // X
    obj1.public_v = … // O
 }
};
```

private/public member variable/function

아무것도 지정하지 않으면 디폴트로 private

```
class Car {
    int private_v;
public:
    int public_v ;
private:
    void private_f();
public:
    void public_f() ;
};
```

함수 in C++
- class member function
- global function (non-member function)

Car class 기준으로 볼 때 함수 in C++
- member function
  - -- of Car class — private member
  - -- of other classes
- global function
  - public member

```
void Car::public_f(){
    private_v = 1;
    public_v = 2;
    private_f();
    public_f();

    Car car3;

    car3.private_v = 3;
    car3.public_v = 4;
    car3.private_f();
    car3.public_f();
}
```

내부

```
class Other {
public:
    void public_f();
};
```

```
void Other::public_f(){
    Car car1;

    car1.private_v = 5; // compile error
    car1.public_v = 6;
    car1.private_f(); // compile error
    car1.public_f();
}
```

외부

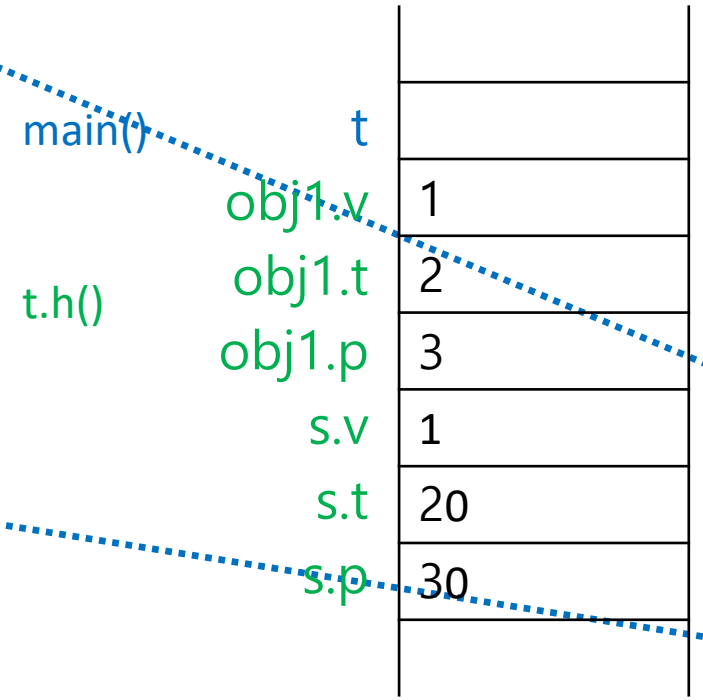
```
int main(){
    Car car2;

    car2.private_v = 7; // compile error
    car2.public_v = 8;
    car2.private_f(); // compile error
    car2.public_f();
}
```

상속에서의 private/public member variable/function

```
4  class A{
5              int v;
6  protected:  int t;
7  public:     int p;
8  void f();
9  };
10 void A::f(){
11  v = 1;
12  t = 2;
13  p = 3;
14 }
15 class SubA: public A{
16 public:
17 void g();
18 };
19 void SubA::g(){
20 // v = 10; // compile error
21  t = 20;
22  p = 30;
23 }
```

A class 기준으로 볼 때 함수 in C++

- member function
  - -- of A class — private member
  - -- of SubA class — protected member
  - -- of other classes
- global function — public member




```
24 class Other{
25 public:
26 void h();
27 };
28 void Other::h(){
29  A obj1;
30 // obj1.v = 100; // compile error
31 // obj1.t = 200; // compile error
32  obj1.p = 300;
33  obj1.f();
34  SubA  s;
35  s.f();
36  s.g();
37 }
```

```
int main(){
  Other t;
  t.h();
}
```

```
(gdb) b h
Breakpoint 1 at 0x804: file inherit.cpp, line 28.
(gdb) b f
Breakpoint 2 at 0x7b2: file inherit.cpp, line 11.
(gdb) b g
Breakpoint 3 at 0x7de: file inherit.cpp, line 21.
(gdb) r
Starting program: /home/ejim/C2020/inherit

Breakpoint 1, Other::h (this=0x7fffffffde97) at inherit.cpp:28
28      void Other::h(){
(gdb) bt full
#0  Other::h (this=0x7fffffffde97) at inherit.cpp:28
        obj1 = {v = 2, t = 0, p = 1431652685}
        s = {<A> = {v = 21845, t = -136423008, p = 32767}, <No data fields>}
#1  0x000055555554878 in main () at inherit.cpp:41
        t = {<No data fields>}
```

local variables 의 값을 function stack 과 함께 보여줌

```
int main(){
  Other t;
  t.h();
}
```

```
24  class Other{
25  public:
26  void h();
27  };
28  void Other::h(){
29   A obj1;
30  // obj1.v = 100;
31  // obj1.t = 200;
32   obj1.p = 300;
33   obj1.f();
34   SubA  s;
35   s.f();
36   s.g();
37  }
```

```
4  class A{
5              int v;
6  protected:  int t;
7  public:     int p;
8  void f();
9  };
10 void A::f(){
11  v = 1;
12  t = 2;
13  p = 3;
14 }
15 class SubA: public A{
16 public:
17 void g();
18 };
19 void SubA::g(
20 // v = 10;
21  t = 20;
22  p = 30;
23 }
```

```
24 class Other{
25 public:
26 void h();
27 };
28 void Other::h(){
29  A obj1;
30 // obj1.v = 100;
31 // obj1.t = 200;
32  obj1.p = 300;
33  obj1.f();
34  SubA  s;
35  s.f();
36  s.g();
```

```
int main(){
  Other t;
  t.h();
}
```

```
(gdb) c
Continuing.

Breakpoint 2, A::f (this=0x7fffffffde60) at inherit.cpp:11
11        v = 1;
(gdb) bt full
#0  A::f (this=0x7fffffffde60) at inherit.cpp:11
No locals.
#1  0x0000555555554826 in Other::h (this=0x7fffffffde97) at inherit.cpp:33
        obj1 = {v = 2, t = 0, p = 300}
        s = {<A> = {v = 21845, t = -136423008, p = 32767}, <No data fields>}
#2  0x0000555555554878 in main () at inherit.cpp:41
        t = {<No data fields>}
```

```
4  class A{
5              int v;
6  protected: int t;
7  public:    int p;
8  void f();
9  };
10 void A::f(){
11  v = 1;
12  t = 2;
13  p = 3;
14 }
15 class SubA: public A{
16 public:
17 void g();
18 };
19 void SubA::g(){
20 // v = 10;
21  t = 20;
22  p = 30;
23 }
```

```
24 class Other{
25 public:
26 void h();
27 };
28 void Other::h(){
29  A obj1;
30 // obj1.v = 100;
31 // obj1.t = 200;
32  obj1.p = 300;
33  obj1.f();
34  SubA  s;
35  s.f();
36  s.g();
37 }
```

```
int main(){
  Other t;
  t.h();
}
```

```
(gdb) c
Continuing.

Breakpoint 2, A::f (this=0x7fffffffde6c) at inherit.cpp:11
11          v = 1;
(gdb) bt full
#0  A::f (this=0x7fffffffde6c) at inherit.cpp:11
No locals.
#1  0x0000555555554832 in Other::h (this=0x7fffffffde97) at inherit.cpp:35
        obj1 = {v = 1, t = 2, p = 3}
        s = {<A> = {v = 21845, t = -136423008, p = 32767}, <No data fields>}
#2  0x0000555555554878 in main () at inherit.cpp:41
        t = {<No data fields>}
```

```
4  class A{
5            int v;
6  protected: int t;
7  public:    int p;
8  void f();
9  };
10 void A::f(){
11  v = 1;
12  t = 2;
13  p = 3;
14 }
15 class SubA: public A{
16 public:
17 void g();
18 };
19 void SubA::g(){
20 // v = 10;
21  t = 20;
22  p = 30;
23 }
```

```
24 class Other{
25 public:
26 void h();
27 };
28 void Other::h(){
29  A obj1;
30 // obj1.v = 100;
31 // obj1.t = 200;
32  obj1.p = 300;
33  obj1.f();
34  SubA  s;
35  s.f();
36  s.g();
37 }
```

```
int main(){
  Other t;
  t.h();
}
```

```
(gdb) c
Continuing.

Breakpoint 3, SubA::g (this=0x7fffffffde6c) at inherit.cpp:21
21        t = 20;
(gdb) bt full
#0  SubA::g (this=0x7fffffffde6c) at inherit.cpp:21
No locals.
#1  0x000055555555483e in Other::h (this=0x7fffffffde97) at inherit.cpp:36
        obj1 = {v = 1, t = 2, p = 3}
        s = {<A> = {v = 1, t = 2, p = 3}, <No data fields>}
#2  0x0000555555554878 in main () at inherit.cpp:41
        t = {<No data fields>}
```

```
4  class A{
5            int v;
6  protected: int t;
7  public:    int p;
8  void f();
9  };
10 void A::f(){
11  v = 1;
12  t = 2;
13  p = 3;
14 }
15 class SubA: public A{
16 public:
17 void g();
18 };
19 void SubA::g(){
20 // v = 10;
21  t = 20;
22  p = 30;
23 }
```

```
24 class Other{
25 public:
26 void h();
27 };
28 void Other::h(){
29  A obj1;
30 // obj1.v = 100;
31 // obj1.t = 200;
32  obj1.p = 300;
33  obj1.f();
34  SubA  s;
35  s.f();
36  s.g();
37 }
```

```
int main(){
  Other t;
  t.h();
}
```

```
(gdb) c
Continuing.

Breakpoint 3, SubA::g (this=0x7fffffffde6c) at inherit.cpp:21
21        t = 20;
(gdb) bt full
#0  SubA::g (this=0x7fffffffde6c) at inherit.cpp:21
No locals.
#1  0x000055555555483e in Other::h (this=0x7fffffffde97) at inherit.cpp:36
        obj1 = {v = 1, t = 2, p = 3}
        s = {<A> = {v = 1, t = 2, p = 3}, <No data fields>}
#2  0x0000555555554878 in main () at inherit.cpp:41
        t = {<No data fields>}
```

```
4  class A{
5            int v;
6  protected: int t;
7  public:    int p;
8  void f();
9  };
10 void A::f(){
11  v = 1;
12  t = 2;
13  p = 3;
14 }
15 class SubA: public A{
16 public:
17 void g();
18 };
19 void SubA::g(){
20 // v = 10;
21  t = 20;
22  p = 30;
23 }
```

```
24 class Other{
25 public:
26 void h();
27 };
28 void Other::h(){
29  A obj1;
30 // obj1.v = 100;
31 // obj1.t = 200;
32  obj1.p = 300;
33  obj1.f();
34  SubA  s;
35  s.f();
36  s.g();
37 }
```

```
int main(){
  Other t;
  t.h();
}
```

```
(gdb) b 37
Breakpoint 4 at 0x55555555483e: file inherit.cpp, line 37.
(gdb) c
Continuing.

Breakpoint 4, Other::h (this=0x7fffffffde97) at inherit.cpp:37
37      }
(gdb) bt full
#0  Other::h (this=0x7fffffffde97) at inherit.cpp:37
        obj1 = {v = 1, t = 2, p = 3}
        s = {<A> = {v = 1, t = 20, p = 30}, <No data fields>}
#1  0x0000555555554878 in main () at inherit.cpp:41
        t = {<No data fields>}
```

```
#include <iostream>
#include <string>

using namespace std;
class  Employee {
      int rrn;
protected:
      int salary;
public:
      string name;
      void setSalary(int salary);
      int getSalary();
};
void Employee::setSalary(int salary) {
      this->salary = salary;
}
int Employee::getSalary() {
      return salary;
}
```

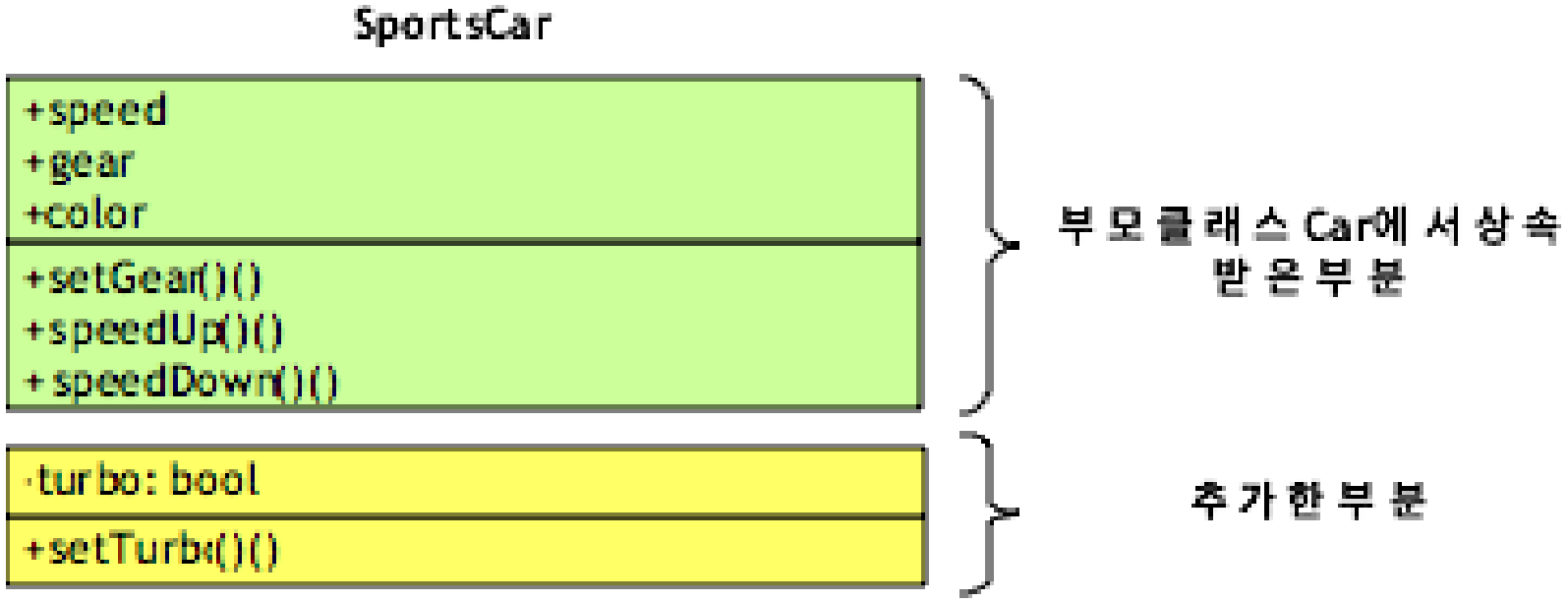
```
class Manager : public Employee {
      int bonus;
public:
      Manager(int b=0) : bonus(b) { }
      void modify(int s, int b);
      void display();
};
void Manager::modify(int s, int b) {
      salary = s;    // 수퍼 클래스의 protected 멤버 사용 가능!
      bonus = b;
}
void Manager::display()
{
      cout << "봉급: " << salary << " 보너스: " << bonus << endl;
      // cout << "주민등록번호: " << rrn << endl;
      // 수퍼 클래스의 private 멤버는 사용할 수 없음!!
}
```

```
int main()
{
      Manager m;
      m.setSalary(2000);   // 수퍼 클래스의 public 함수 사용
      m.display();
      m.modify(1000, 500);
      m.display();
}
```

```
봉급: 1000 보너스: 500
계속하려면 아무 키나 누르십시오 . . .
```

상속에서의 생성자와 소멸자

- 서브 클래스의 객체가 생성될 때에 당연히 서브 클래스의 생성자는 호출된다. 이때 수퍼 클래스 생성자도 호출될까?



상속에서의 생성자와 소멸자

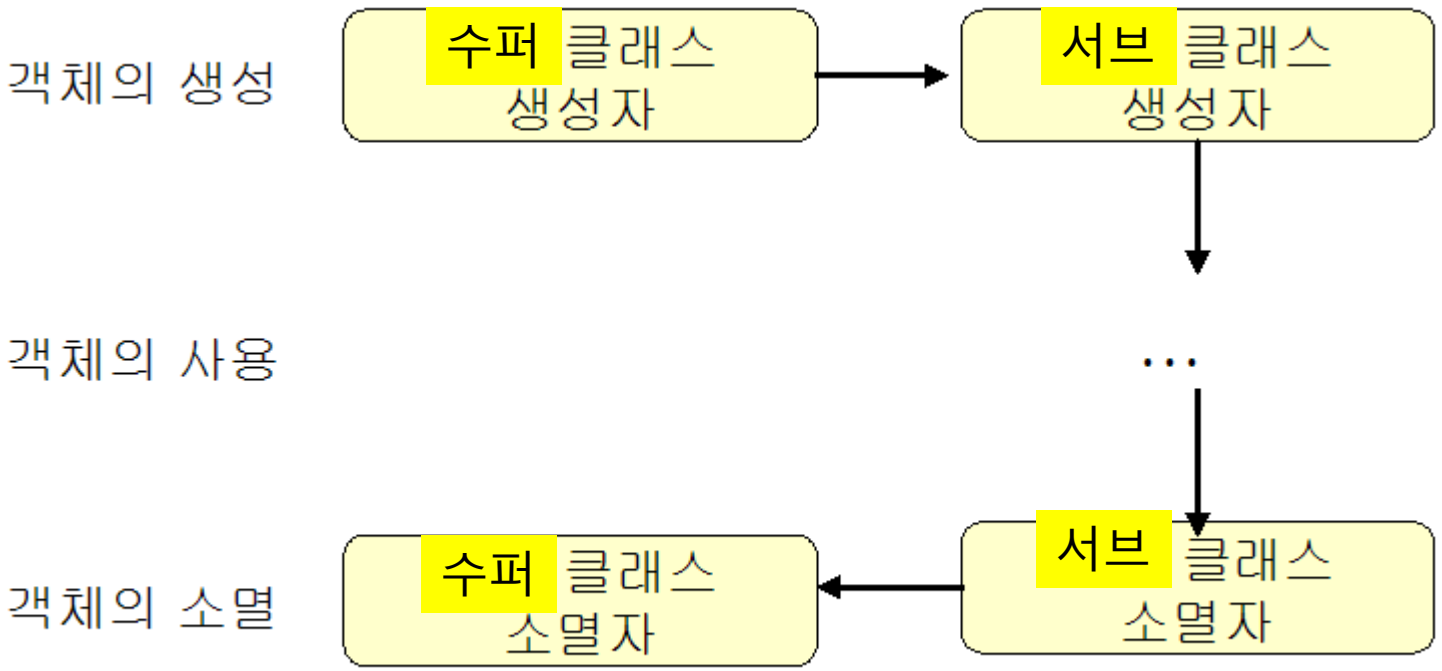


그림 13.10 상속에서 생성자와 소멸자의 호출

```
class Shape{
  int x,y;
public:
  Shape(int x=0,int y=0):x(x), y(y){
    cout << this << " Shape()" << endl;
  }
  ~Shape(){
    cout << this << " ~Shape()" << endl;
  }
  void print(){
     cout << "( " << x << ", " << y << " ) ";
  }
};
```

```
class Rectangle : public Shape{
  int width, height;
public:
  Rectangle(int x=0, int y=0, int w=0, int h=0){
    width = w; height = h;
    cout << this << " Rectangle()" << endl;
  }
  ~Rectangle(){
    cout << this << " ~Rectangle()" << endl;
  }
  void print(){
    Shape::print();
    cout << " : " << width << " x " << height << endl;
  }
};
```

Shape() default constructor 가 먼저 실행

```
int main(){
  Rectangle r;
  return 0;
}
```

```
0x7ffdfe527bd0 Shape()
0x7ffdfe527bd0 Rectangle()
0x7ffdfe527bd0 ~Rectangle()
0x7ffdfe527bd0 ~Shape()
```

```cpp
class Shape{
  int x,y;
public:
  Shape(int x=0,int y=0):x(x), y(y){
    cout << this << " Shape()" << endl;
  }
  ~Shape(){
    cout << this << " ~Shape()" << endl;
  }
  void print(){
     cout << "( " << x << ", " << y << " ) ";
  }
};
```

```cpp
class Rectangle : public Shape{
  int width, height;
public:
  Rectangle(int x=0, int y=0, int w=0, int h=0): Shape(x,y){
    width = w; height = h;
    cout << this << " Rectangle()" << endl;
  }
  ~Rectangle(){
    cout << this << " ~Rectangle()" << endl;
  }
  void print(){
    Shape::print();
    cout << " : " << width << " x " << height << endl;
  }
};
```

```cpp
int main(){
  Rectangle r;
  return 0;
}
```

```
0x7ffdfe527bd0 Shape()
0x7ffdfe527bd0 Rectangle()
0x7ffdfe527bd0 ~Rectangle()
0x7ffdfe527bd0 ~Shape()
```

수퍼 클래스 생성자의 명시적 호출

```
Rectangle(int x=0, int y=0, int w=0, int h=0): Shape(x,y){
  width = w; height = h;
  cout << this << " Rectangle()" << endl;
}
```

클래스 이름

(비교) 객체 멤버의 경우 : Has-A relationship

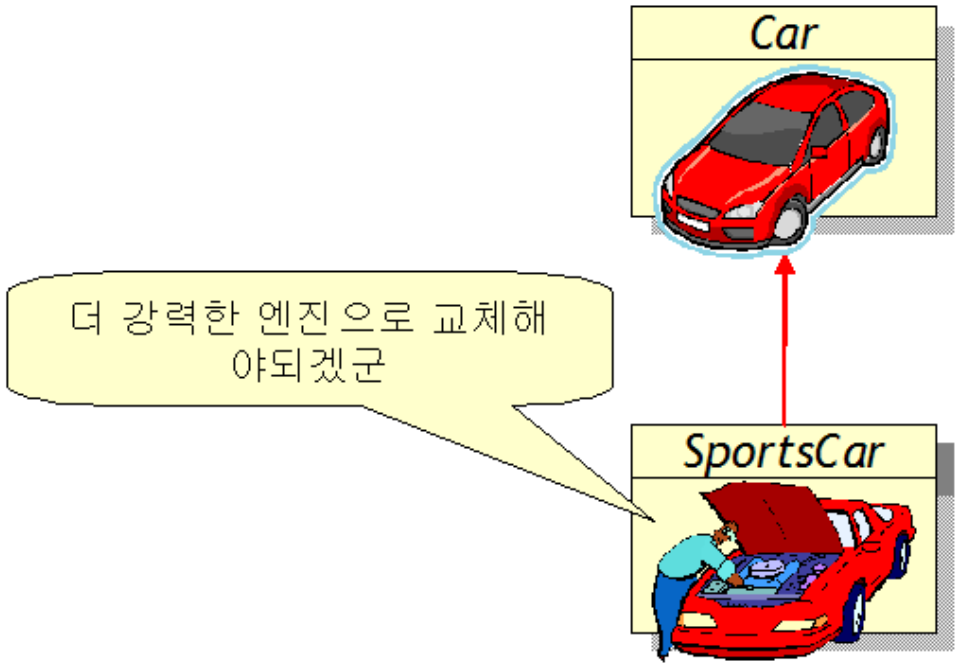
```
4  class Point{
5    int x,y;
6  public:
7    Point(int x,int y):x(x){
8       this->y = y;
9    }
10 };
11 class Circle{
12   Point center;
13   int radius;
14 public:
15   Circle(int x,int y, int r):center(x,y), radius(r){}
16 };
17 int main(){
18   Circle c1(4,5,2);
19   return 0;
20 }
```

멤버 객체의 생성자 호출

멤버 변수 이름

멤버 함수의 재정의 Overriding

- 재정의: 서브클래스가 필요에 따라 상속된 멤버 함수를 다시 정의하는 것



```
4  class Car{
5  public:
6    int getHP(){ return 100; }
7  };
8  class SportsCar : public Car {
9  public:
10   int getHP(){ return 300; }
11 };
12
13 int main(){
14   SportsCar c;
15
16   cout << "HorsePower : " << c.getHP() << endl;
17   cout << "HorsePower : " << c.Car::getHP() << endl;
18   return 0;
19 }
```

main() 에서는
Car::getHP(); 호출할 수 없음
static member function 이 아님

```
HorsePower : 300
HorsePower : 100
```

멤버 변수의 재정의 Overriding

```
4  class Car{
5  public:
6    int speed;
7    Car() : speed(0){}
8    int getHP(){ return 100; }
9  };
10 class SportsCar : public Car {
11 public:
12   int speed;
13   SportsCar(): speed(1){}
14   int getHP(){ return 300; }
15 };
16
17 int main(){
18   SportsCar c;
19   cout << "HorsePower : " << c.getHP() << endl;
20   cout << "HorsePower of Superclass : " << c.Car::getHP() << endl;
21   cout << "Speed : " << c.speed << endl;
22   cout << "Speed of Superclass : " << c.Car::speed << endl;
23   return 0;
24 }
```

```
HorsePower : 300
HorsePower of Superclass : 100
Speed : 1
Speed of Superclass : 0
```

재정의의 조건

- 부모 클래스의 멤버 함수와 동일한 시그니처를 가져야 한다.
- 즉 멤버 함수의 이름, 반환형, 매개 변수의 개수와 데이터 타입이 일치하여야 한다.

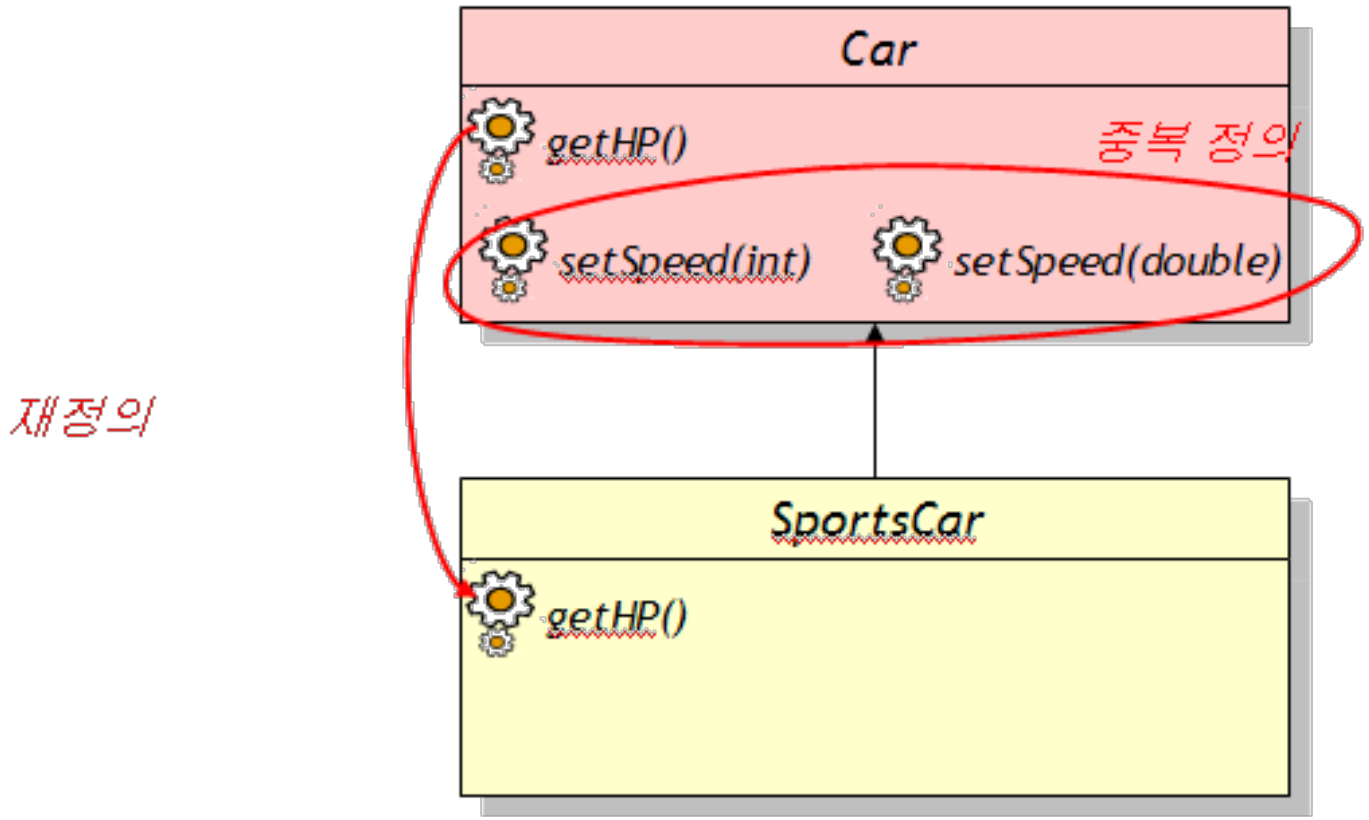
```
class Animal {
        void makeSound()
        {
        }
};
```

재정의가 아님

```
class Dog : public Animal {
        void makeSound(string s)
        {
        }
};
```

재정의(overriding)와 중복 정의(overloading)

- 중복 정의: 같은 이름의 멤버 함수를 여러 개 정의하는 것
- 재정의: 부모 클래스에 있던 상속받은 멤버 함수를 다시 정의하는 것



재정의된 멤버 함수의 호출 순서

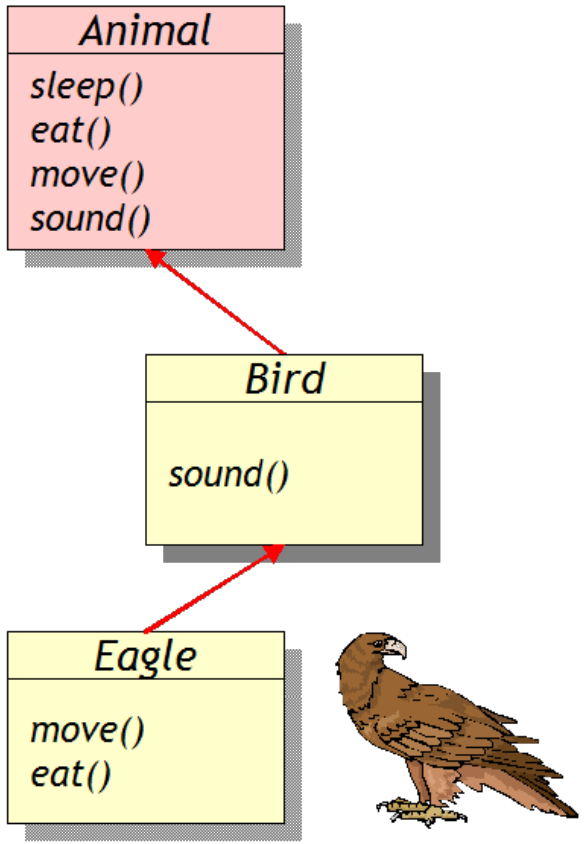


그림 13.13 상속 계층 구조

```
Eagle e;
e.sleep();// Animal의 sleep() 호출
e.eat();// Eagle의 eat() 호출
e.sound();// Bird의 sound() 호출
```

서브클래스의 멤버함수에서 수퍼클래스의 멤버함수 호출

```
4  class Car{
5  public:
6    int speed;
7    Car() : speed(0){}
8    int getHP(){ return 100; }
9    void print(){ cout << "Car\n"; }
10 };
11 class SportsCar : public Car {
12 public:
13   int speed;
14   SportsCar(): speed(1){}
15   int getHP(){ return 300; }
16   void print(){
17     Car::print();
18     cout << "SportsCar\n";
19   }
20 };
21
22 int main(){
23   SportsCar c;
24   c.print();
25 }
```

```
Car
SportsCar
```

부모 클래스의 함수 호출!

main() 에서는
Car::print(); 호출할 수 없음
static member function 이 아님

(상속을 고려한) class A 의 멤버에 대한 접근 제어 지정자

접근 지정자	현재 클래스 A 의 멤버 함수	서브클래스 subA 의 멤버 함수	클래스 B 의 멤버 함수/전역 함수
private	O	X	X
protected	O	O	X
public	O	O	O

```
class A {
  private : int private_v;
  protected : int protected_v;
  public: int public_v;
   void ftn1 (){
      private_v = …      // O
      protected_v = …   // O
      public_v = …       // O
   }
};
```

```
class subA : p..... A {
 void ftn2(){
   private_v = … // X
   protected_v = … // O
   public_v = … // O
 }
};
```

```
class B{
  void ftn3(){
    A  obj1;
    obj1.private_v =… // X
    obj1.protected_v =… // X
    obj1.public_v = … // O
  }
};
```

상속의 3가지 유형에 따른

서브클래스의 멤버 함수에서 접근 가능성

	public으로 상속하면	protected로 상속하면	private로 상속하면
수퍼클래스의 private 멤버	접근 안됨	접근 안됨	접근 안됨
수퍼클래스의 protected 멤버	->protected	->protected	->private
수퍼클래스의 public 멤버	->public	->protected	->private

상속의 3가지 유형에 따른

서브클래스의 멤버 함수에서 접근 가능성

```
class A {
  private : int private_v;
  protected : int protected_v;
  public: int public_v;
};
```

	public으로 상속하면	protected로 상속하면	private로 상속하면
수퍼클래스의 private 멤버	접근 안됨	접근 안됨	접근 안됨
수퍼클래스의 protected 멤버	->protected	->protected	->private
수퍼클래스의 public 멤버	->public	->protected	->private

집 안

```
A?::g() {
  private_v = ...    // X
  protected_v = ...// O
  public_v = ...     // O
}
```

A1 or A2 or A3 class 기준

집 밖1 (subclass의 멤버함수)

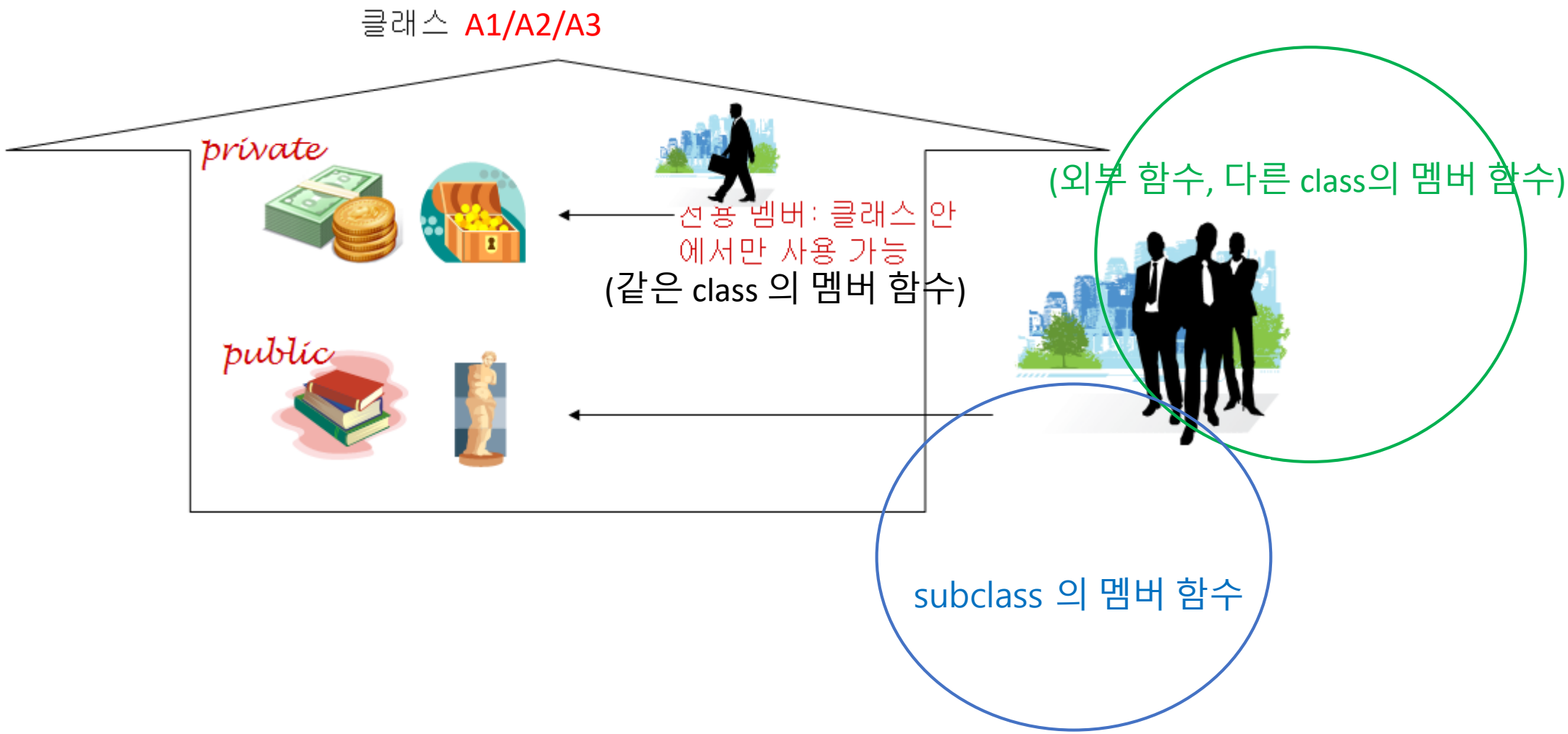
```
class A1  : public A { void g(); };
class A11 : p.... A1 { // subclass
  void f1(){
    private_v = ...    // X
    protected_v = ...// O
    public_v = ...     // O
  }
};
int main(){          // 외부 함수
  A1 a1;
  a1.private_v = ... // X
  a1.protected_v = ... // X
  a1.public_v = ... // O
```

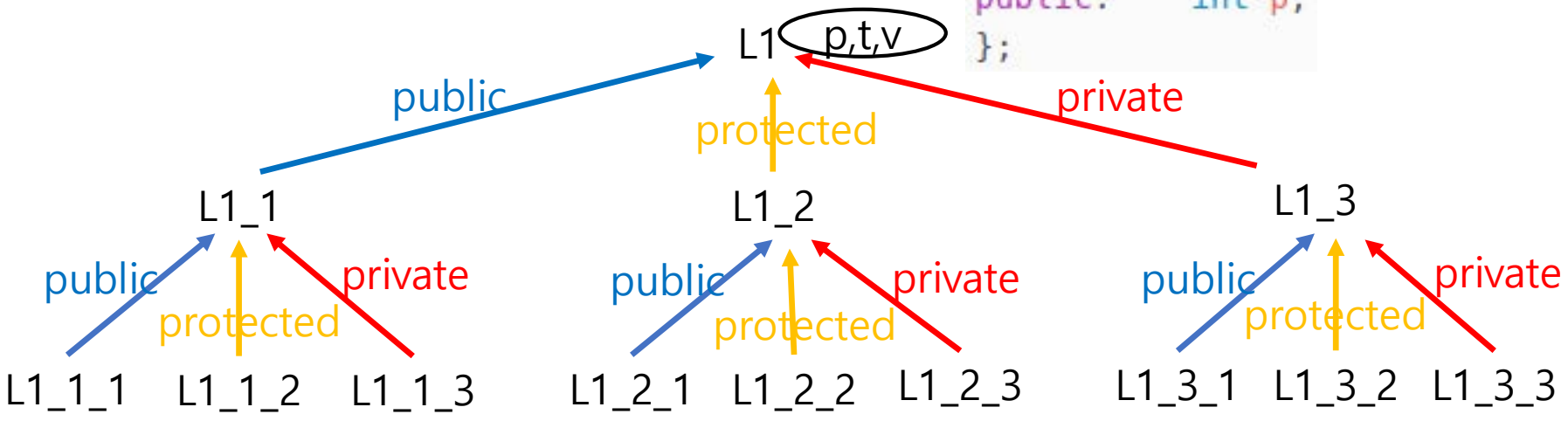
```
class A2 :  protected A { void g(); };
class A21 : p.... A2{
  void f2(){
    private_v = ...    // X
    protected_v = ...// O
    public_v = ...     // O
  }
};
int main(){
  A2 a2;
  a2.private_v = ... // X
  a2.protected_v = ... // X
  a2.public_v = ... // X
```

```
class A3 :  private A { void g(); };
class A31 : p.... A3 {
  void f3(){
    private_v = ...    // X
    protected_v = ...// X
    public_v = ...     // X
  }
};
int main(){
  A3 a3;
  a3.private_v = ... // X
  a3.protected_v = ... // X
  a3.public_v = ... // X
```

집 밖2 (다른 클래스의 멤버함수, 전역 함수)

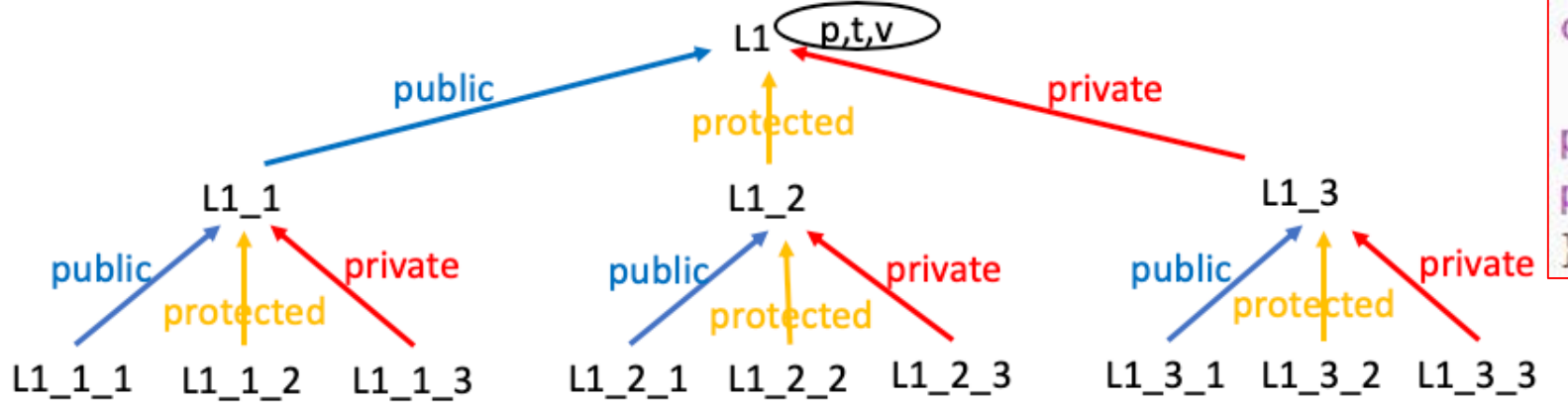
상속을 고려한 접근 제어자 private – protected - public






```
class L1{
            int v;
protected:  int t;
public:     int p;
};
```

```
class L1_1: public L1{
public:
  void g(){
// v = 11;
   t = 11; //->protected
   p = 11; //->public
  }
};
class L1_2: protected L1{
public:
  void g(){
// v = 12;
   t = 12; //->protected
   p = 12; //->protected
  }
};
class L1_3: private L1{
public:
  void g(){
// v = 13;
   t = 13; //->private
   p = 13; //->private
  }
};
```



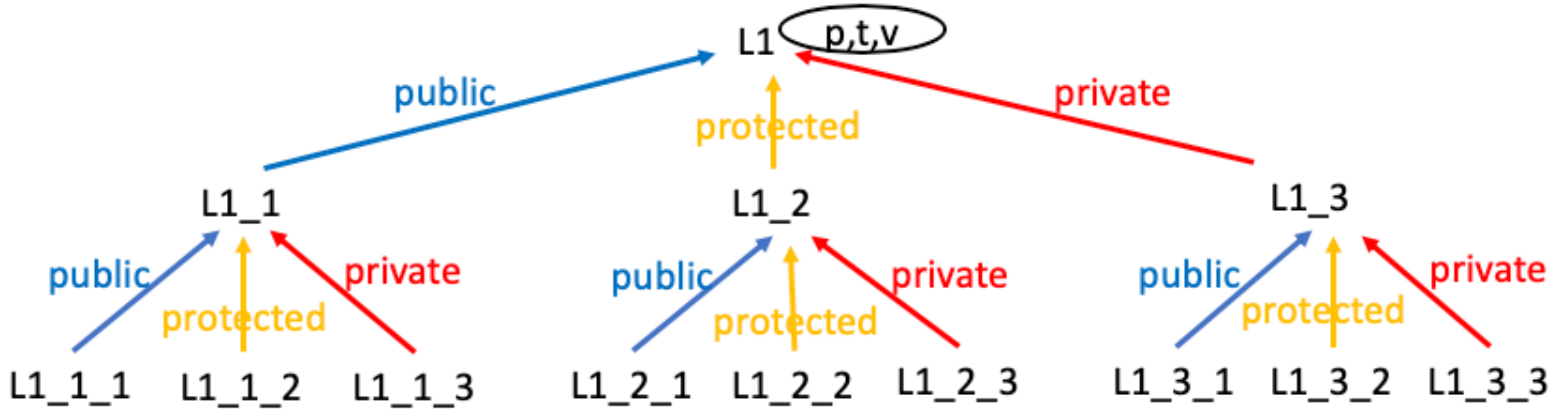

```
class L1{
                int v;
protected:      int t;
public:         int p;
};
```

```
class L1_1_1: public L1_1{
public:
  void f(){
// v = 111;
   t = 111; //->protected
   p = 111; //->public
  }
};
class L1_1_2: protected L1_1{
public:
  void f(){
// v = 112;
   t = 112; //->protected
   p = 112; //->protected
  }
};
class L1_1_3: private L1_1{
public:
  void f(){
// v = 113;
   t = 113; //->private
   p = 113; //->private
  }
};
```

```
class L1_2_1: public L1_2{
public:
  void f(){
// v = 121;
   t = 121; //->protected
   p = 121; //->protected
  }
};
class L1_2_2: protected L1_2{
public:
  void f(){
// v = 122;
   t = 122; //->protected
   p = 122; //->protected
  }
};
class L1_2_3: private L1_2{
public:
  void f(){
// v = 123;
   t = 123; //->private
   p = 123; //->private
  }
};
```

```
class L1_3_1: public L1_3{
public:
  void f(){
// v = 131;
// t = 131;
// p = 131;
  }
};
class L1_3_2: protected L1_3{
public:
  void f(){
// v = 132;
// t = 132;
// p = 132;
  }
};
class L1_3_3: private L1_3{
public:
  void f(){
// v = 133;
// t = 133;
// p = 133;
  }
};
```

```
class L1_1: public L1{
public:
  void g(){
// v = 11;
   t = 11; //->protected
   p = 11; //->public
  }
};
class L1_2: protected L1{
public:
  void g(){
// v = 12;
   t = 12; //->protected
   p = 12; //->protected
  }
};
class L1_3: private L1{
public:
  void g(){
// v = 13;
   t = 13; //->private
   p = 13; //->private
  }
};
```




```
109  void Other::h(){
110   L1 obj1;
111   L1_1 obj1_1;
112   L1_2 obj1_2;
113   L1_3 obj1_3;
114   L1_1_1 obj1_1_1;
115   L1_1_2 obj1_1_2;
116   L1_1_3 obj1_1_3;
117   L1_2_1 obj1_2_1;
118   L1_2_2 obj1_2_2;
119   L1_2_3 obj1_2_3;
120   L1_3_1 obj1_3_1;
121   L1_3_2 obj1_3_2;
122   L1_3_3 obj1_3_3;
```

```
123  // obj1.v = 1000; private
124  // obj1.t = 1000; protected
125   obj1.p = 1000;
126  // obj1_1.v = 1000;
127  // obj1_1.t = 1000; protected
128   obj1_1.p = 1000;
129  // obj1_2.v = 1000;
130  // obj1_2.t = 1000; protected
131  // obj1_2.p = 1000; protected
132  // obj1_3.v = 1000;
133  // obj1_3.t = 1000; private
134  // obj1_3.p = 1000; private
135  // obj1_1_1.v = 1000;
136  // obj1_1_1.t = 1000; protected
137   obj1_1_1.p = 1000;
138  // obj1_1_2.v = 1000;
139  // obj1_1_2.t = 1000; protected
140  // obj1_1_2.p = 1000; protected
141  // obj1_1_3.v = 1000;
142  // obj1_1_3.t = 1000; private
143  // obj1_1_3.p = 1000; private
144  }
```

예제

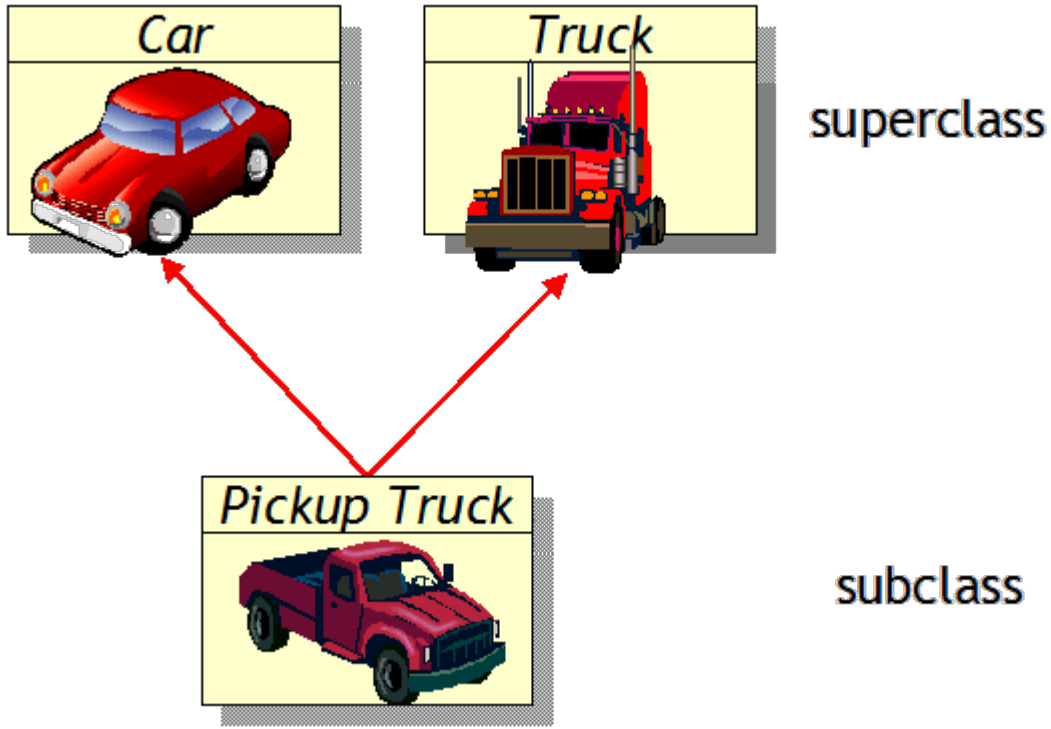
```
#include <iostream>
using namespace std;

class ParentClass {
public:
      const static int x=100;          // 정적 상수 정의는 초기화가 가능하다.
};

class ChildClass1 : public ParentClass {
};
class ChildClass2 : private ParentClass {
};

int main()
{
   ChildClass1 obj1;
   ChildClass2 obj2;
   cout << obj1.x << endl;          // 가능: x는 public으로 유지된다.
   cout << obj2.x << endl;          // 오류!!! 불가능: x는 public에서 private로 변경되었다.
   return 0;
}
```

예제 - 계속

```
class ParentClass {
public:
      const static int x=100;          // 정적 상수 정의는 초기화가 가능하다.
};

class ChildClass1 : public ParentClass {
   f() { cout<< x << endl;   // O
};
class ChildClass2 : private ParentClass {
   f() { cout<< x << endl;   // O
};
class GrandChildClass1_1 : private ChildClass1 {
   f() { cout<< x << endl;   // O
};
class GrandChildClass2_1 : public ChildClass2 {
   f() { cout<< x << endl;   // X
};
```

다중 상속

```
class Sub : public Sup1, public Sup2
{
...// 추가된 멤버
...// 오버라이딩된 멤버
}
```



예제

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

class PassengerCar {
public:
	int seats; // 정원
	void set_seats(int n){ seats = n; }
};

class Truck {
public:
	int payload; // 적재 하중
	void set_payload(int load){ payload = load; }
};

class Pickup : public PassengerCar, public Truck {
public:
	int tow_capability; // 견인 능력
	void set_tow(int capa){ tow_capability = capa; }
};
```

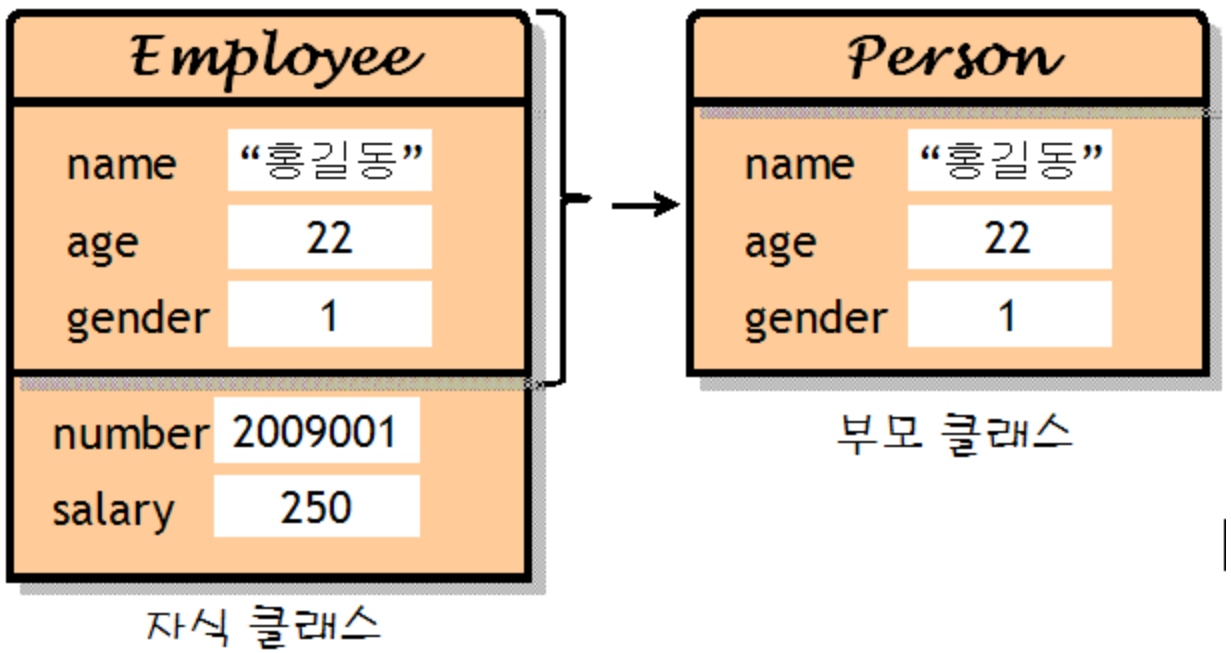
```
int main()
{
	Pickup my_car;
	my_car.set_seats(4);
	my_car.set_payload(10000);
	my_car.set_tow(30000);
	return 0;
}
```

다중 상속의 문제점 : ambiguity

```
class SuperA
{
public:
    int x;
    void sub(){
        cout << "SuperA의 sub()" << endl;
    }
};
class SuperB
{
public:
    int x;
    void sub(){
        cout << "SuperB의 sub()" << endl;
    }
};
```

```
class Sub : public  SuperA, public SuperB
{
};

int main()
{
    Sub obj;
    obj.x = 10;  // obj.x는 어떤 부모 클래스의 x를 참조하는가? obj.SuperA::x
    return 0;
}
```

예제


Employee
name
"홍길동"
age
22
gender
1
number
2009001
salary
250
자식 클래스
Person
name
"홍길동"
age
22
gender
1
부모 클래스

예제

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

class Person {
	string name;
	int age;
	bool gender;
public:
	Person(string n="", int a=0, bool g=true): name(n), age(a), gender(g) { }
	void setName(string s) { name = s; }
	string getName() const { return name; }
	void setAge (int a) { age = a; }
	int getAge() const { return age; }
	void setGender (bool g) { gender = g; }
	bool getGender() const { return gender; }
};
```

예제

```
class Employee : public Person {
        int number;
        int salary;
public:
        Employee(string n="", int a=0, bool g=true, int num=0, int s=0): Person(n,  a, g),
        number(num), salary(s) { }
        void display() const;
        void setNumber (int n) { number = n; }
        int getNumber() const { return number; }
        void setSalary (int s) { salary = s; }
        int getSalary() const { return salary; }
};

void Employee::display() const
{
        cout << this->getName() << endl;
        cout << this->getAge() << endl;
        cout << this->getGender() << endl;
        cout << this->getNumber() << endl;
        cout << this->getSalary() << endl;
}
```

예제

```
int main()
{
	Employee e("김철수", 26, true, 2010001, 2800);
	e.display();
	return 0;
}
```

```
김철수
26
1
2010001
2800
계속하려면 아무 키나 누르십시오 . . .
```

실습

Kvector class 를 상속하여 Avector class 를 만들어라.

- Kvector class 의 멤버 변수는 protected 로 선언하라.
- Avector class 는 table 라는 문자 배열을 member 변수로 가지는데 그 원소들은 각각 0,1,2 의 값에 대응되는 영어 알파벳이다.
- Avector 의 생성자는 table 의 초기값을 문자열 상수 인자로부터 초기화한다.
- Avector 의 member function setTable() 함수는 table 의 내용을 변경한다.
- Avector 의 객체를 출력할 때는 m 배열의 원소들을 3으로 나눈 나머지를 0,1,2 대신에 table 배열에서 이에 해당하는 알파벳이 출력되어야 한다. (<< 연산자를 overriding 하라.)

```
 4  class Kvector{
 5  protected:
 6    int *m;
 7    int len;
 8  public:
 9 ›  Kvector(int sz = 0, int value = 0): len(sz){⋯}
15 ›  Kvector(const Kvector& v){⋯}
22 ›  ~Kvector(){⋯}
26 ›  void print() const {⋯}
30    void clear(){ delete[] m;    m = NULL;    len = 0;}
31    int size(){ return len; }
32  Kvector& operator=(const Kvector& v);
33  friend bool operator==(const Kvector& v, const Kvector& w);
34  friend bool operator!=(const Kvector& v, const Kvector& w);
35  int& operator[](int idx){ return m[idx]; }
36  const int& operator[](int idx) const { return m[idx]; }
37  friend ostream& operator<<(ostream& os, const Kvector& v);
38  };
39 › Kvector& Kvector::operator=(const Kvector& v){⋯}
46 › bool operator==(const Kvector& v, const Kvector& w){⋯}
52 › bool operator!=(const Kvector& v, const Kvector& w){⋯}
55 › ostream& operator<<(ostream& os, const Kvector& v){⋯}
```

```
59  #define N 3
60  class Avector : public Kvector{
61    char table[N];
62  public:
63  ›   Avector(int sz=0, int v=0, const char *t="abc"): Kvector(sz, v){⋯}
67  ›   void setTable(const char *t){⋯}
70  friend ostream& operator<<(ostream& os, const Avector& v);
71  };
72  ostream& operator<<(ostream& os, const Avector& v){⋯}
```

```
77  int main(int argc, char *argv[]){
78    if (argc !=2 ){
79      cout << "usage : ./avecter pqr\n";
80      return 1;
81    }
82    Avector v1(3);  v1.print();
83    Avector v2(2, 1, "xyz");  v2.print();
84    Avector v3(v2);  v3.print();
85    cout << (v1 == v2) << endl;
86    cout << (v3 == v2) << endl;
87    v3 = v2 = v1;
88    cout << v1 << endl;
89    v1.print();
90    cout << v2 << endl;
91    v2.print();
92    cout << v3 << endl;
93    v3.print();
94    cout << (v3 != v2) << endl;
95    v1[2] = 2;
96    v2[0] = v1[2];
97    v1.setTable(argv[1]);
98    cout << "v1: " << v1 << "v2: " << v2 <<"v3: " << v3 <<
99    v1.print();
100   v2.print();
101   v3.print();
102   return 0;
```

```
0x7ffd64b96da0 : Kvector(int, int)
0x7ffd64b96da0 : Avector(int, int, const char*)
abc
0 0 0
0x7ffd64b96db0 : Kvector(int, int)
0x7ffd64b96db0 : Avector(int, int, const char*)
xyz
1 1
0x7ffd64b96dc0 : Kvector(Kvector&)
1 1
0
1
a a a
0 0 0
a a a
0 0 0
a a a
0 0 0
0
v1: p p r v2: c a a v3: a a a
0 0 2
2 0 0
0 0 0
0x7ffd64b96dc0 : ~Kvector()
0x7ffd64b96db0 : ~Kvector()
0x7ffd64b96da0 : ~Kvector()
```