11장 상속

2020. 11. 9

순천향대학교 컴퓨터 공학과

내용

- 상속 개념
- 상속시 생성자와 소멸자 호출 순서
- 부모클래스 생성자 호출
- 상속과 접근 지정자
- 상속시 접근 지정
- 다중 상속

상속의 개요

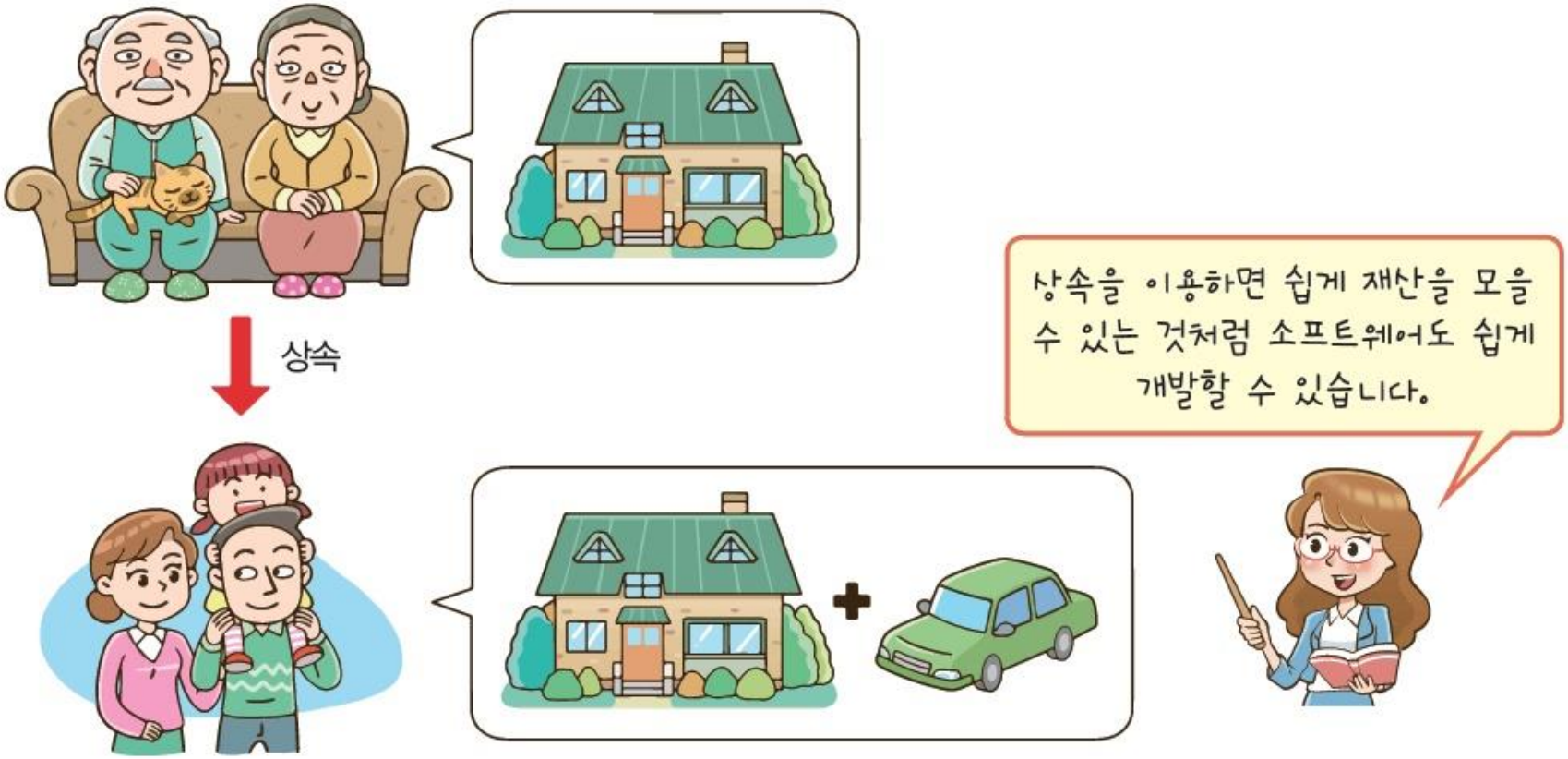


그림 11.1 상속의 개념

C++에서의 상속(Inheritance)

- C++에서의 상속이란?
  - 클래스 사이에서 상속 관계 정의
  - 기본 클래스의 속성과 기능을 파생 클래스에 물려주는 것
    - 기본 클래스(base class) - 상속해주는 클래스(부모 클래스)
    - 파생 클래스(derived class) – 상속받는 클래스(자식 클래스)
    - 기본 클래스의 속성과 기능을 물려받고 자신만의 속성과 기능을 추가하여 작성
  - 기본 클래스에서 파생 클래스로 갈수록 클래스의 개념이 구체화
  - 다중 상속을 통한 클래스의 재활용성 높임

상속의 표현

상속 관계 표현

```
class Phone {
   void call();
   void receive();
};
```

Phone을 상속받는다.

```
class MobilePhone : public Phone {
   void connectWireless();
   void recharge();
};
```

MobilePhone을 상속받는다.

```
class MusicPhone : public MobilePhone {
   void downloadMusic();
   void play();
};
```

C++로 상속 선언

전화기

휴대 전화기

음악 기능 전화기

상속의 목적 및 장점

- 간결한 클래스 작성
  - 기본 클래스의 기능을 물려받아 파생 클래스를 간결하게 작성
- 클래스 간의 계층적 분류 및 관리의 용이함
  - 상속은 클래스들의 구조적 관계 파악 용이
- 소프트웨어 생산성 향상
  - 빠른 소프트웨어 생산 필요
  - 기존에 작성한 클래스의 재사용 – 상속
  - 상속받아 새로운 기능을 확장
  - 객체 지향적 설계 필요

상속 정의 구문

상속 접근 지정자

문법 11.3 다중 상속

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

예제

```
// 한 점을 표현하는 클래스 Point 선언
class Point {
   int x, y; //(x,y) 좌표값
public:
   void set(int x, int y)
  { this->x = x; this->y = y; }
   void showPoint() {
      cout << "(" << x << "," << y << ")" << endl;
   }
};
```

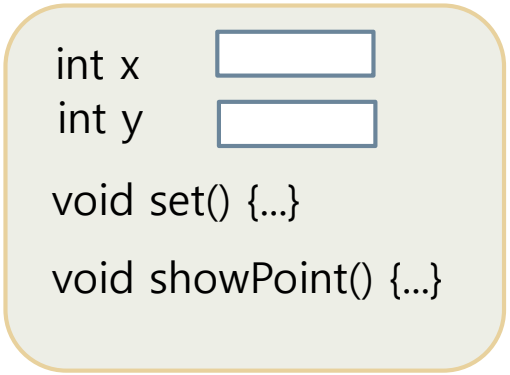
```
class ColorPoint : public Point {
   string color;// 점의 색 표현
public:
   void setColor(string color)
  {  this->color = color; }
   void showColorPoint();
};

void ColorPoint::showColorPoint() {
   cout << color << ":";
   showPoint();
}

int main() {
   Point p;
   ColorPoint cp;

   cp.set(3,4);
   cp.setColor("Red");
   cp.showColorPoint();
}
```

자식 클래스의 객체 구성

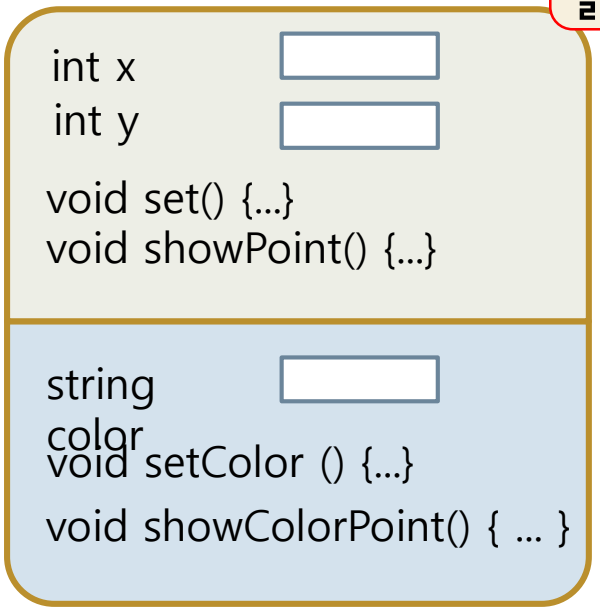
```
class Point {
   int x, y;
public:
   void set(int x, int y);
   void showPoint();
};
```

```
class ColorPoint : public Point {
   string color;
public:
   void setColor(string color);
   void showColorPoint();
};
```

Point p;



ColorPoint cp;

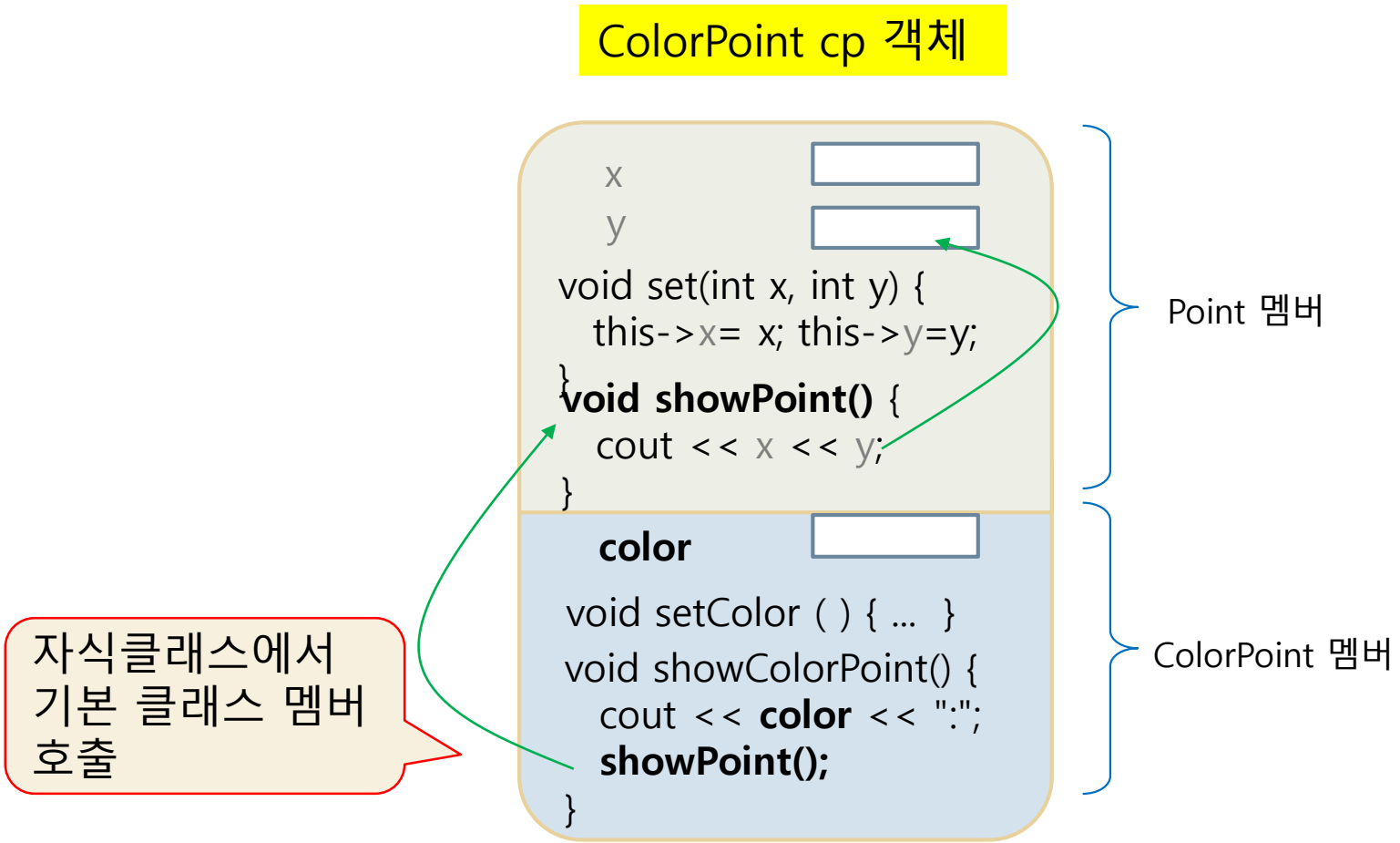


자식클래스의 객체는 기본 클래스의 멤버 포함

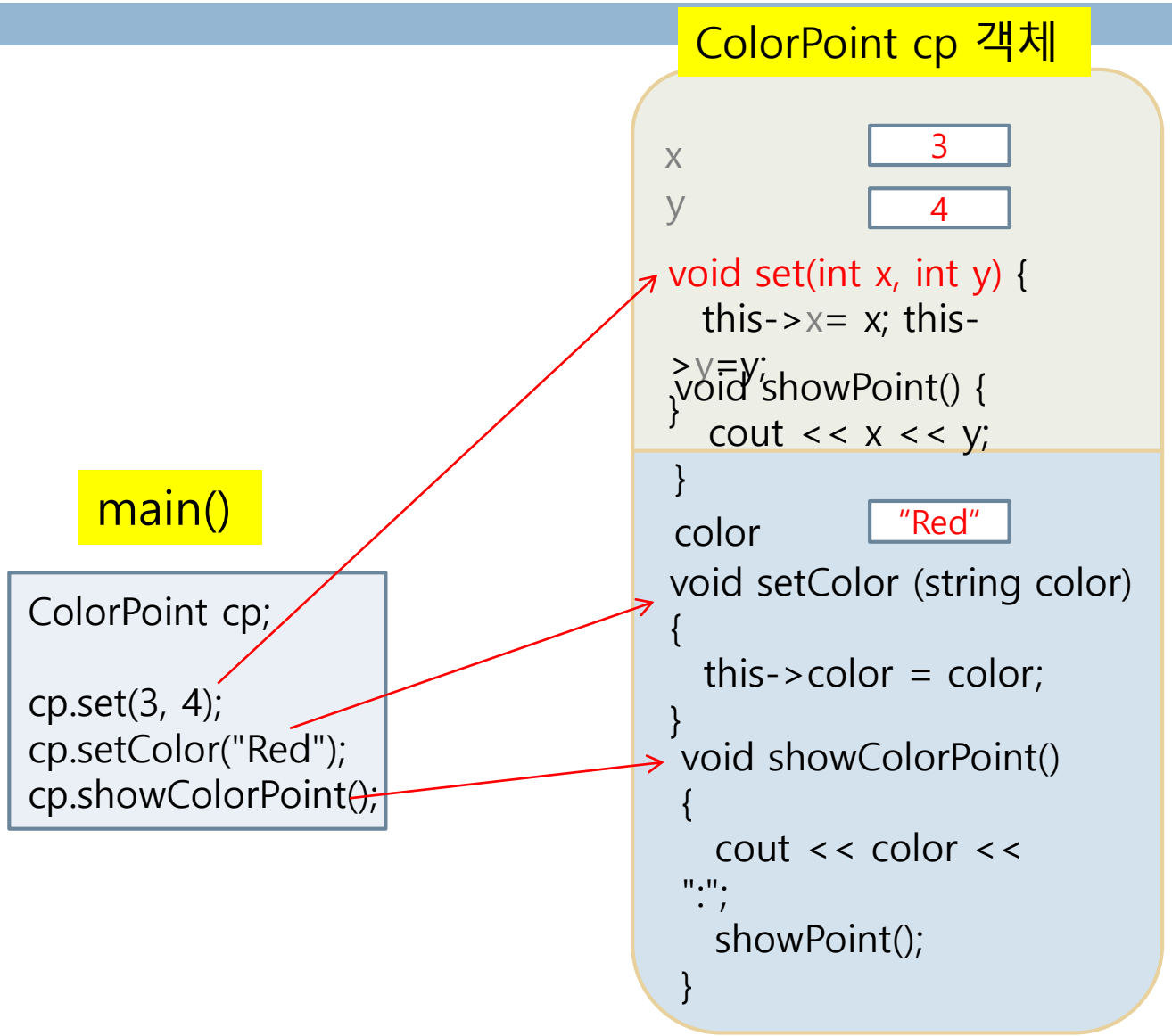
기본클래스 멤버

자식클래스 멤버

자식 클래스에서 기본 클래스 멤버 접근



외부에서 자식 클래스 객체에 대한 접근

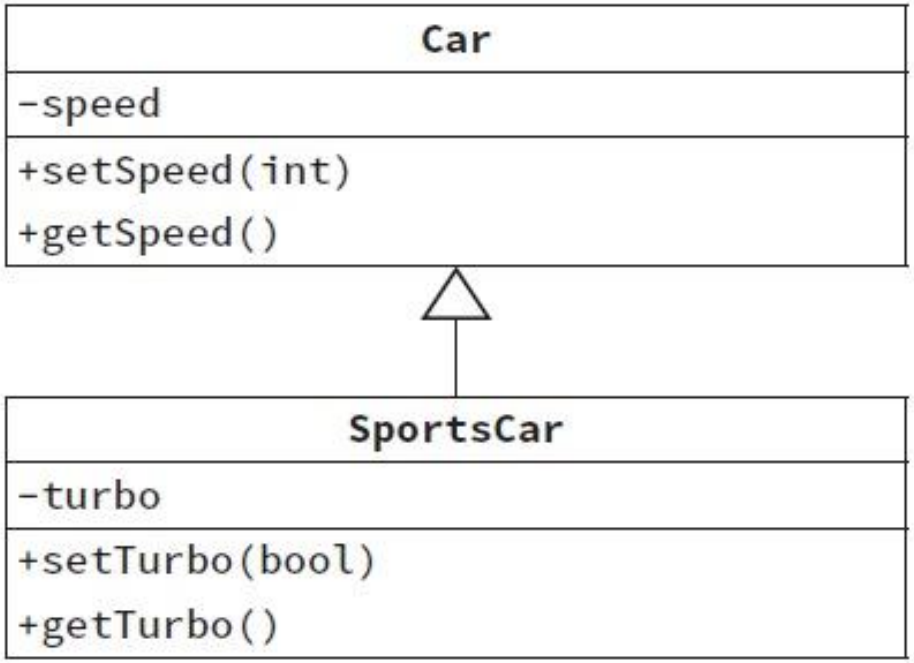


```
class Car {
        int speed; // 속도

public:
        void setSpeed(int s) {
          speed = s;
        }
        int getSpeed() {
          return speed;
        }
};

// Car 클래스를 상속받아서 다음과 같이 SportsCar 클래스를 작성한다.
class SportsCar : public Car {
        bool turbo;
public:
        void setTurbo(bool newValue) {
          turbo = newValue;
        }
        bool getTurbo() {
          return turbo;
        }
};
```

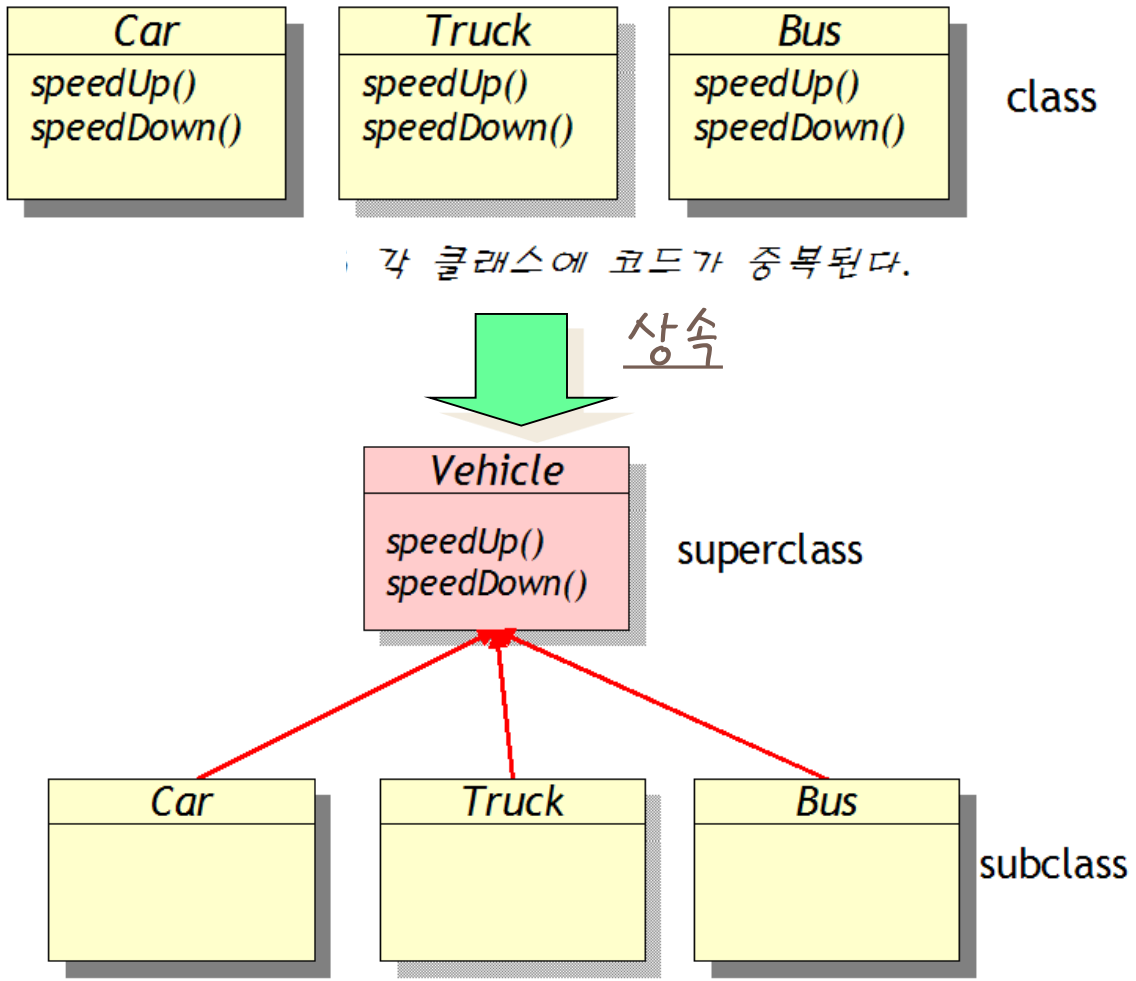
```
int main()
{
        SportsCar  c;

        c.setSpeed(60);
        c.setTurbo(true);
        c.setSpeed(100);
        c.setTurbo(false);
        return 0;
}
```

상속 관계 표현

- UML 이용

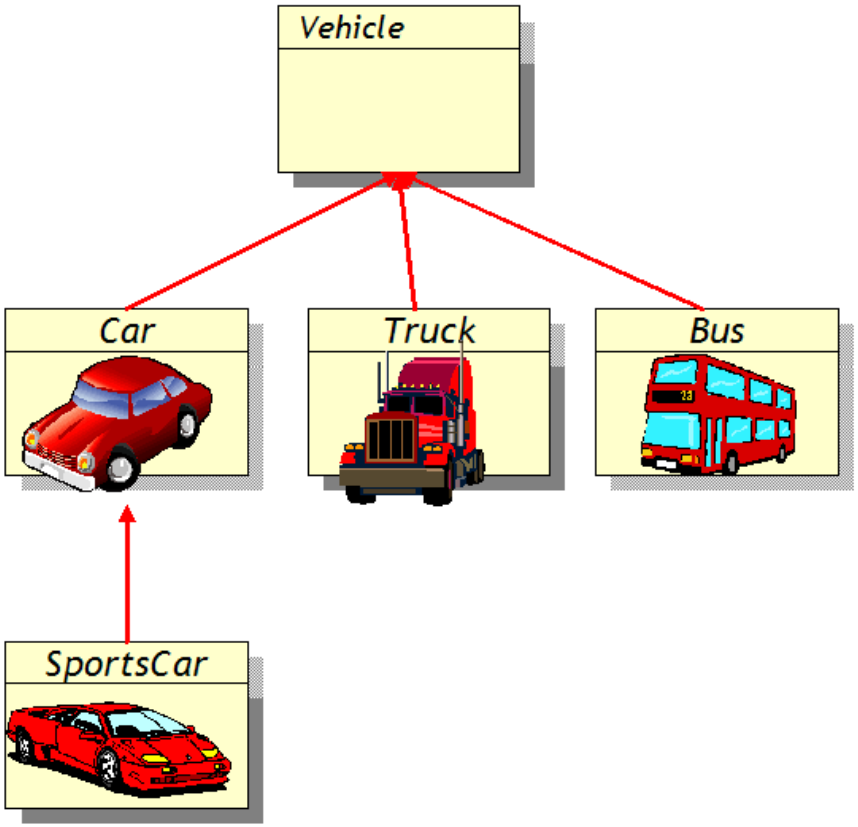


상속은 왜 필요한가?

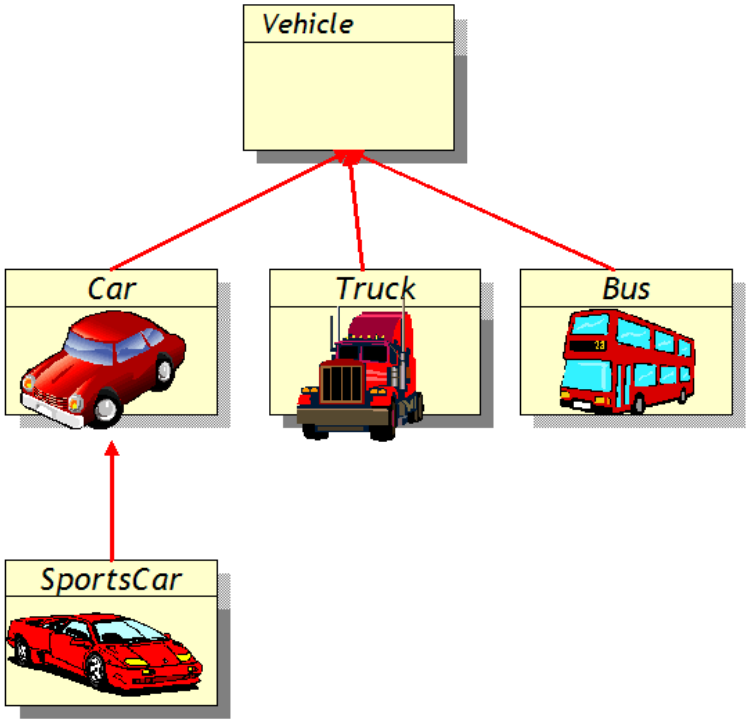

Car
speedUp()
speedDown()
Truck
speedUp()
speedDown()
Bus
speedUp()
speedDown()
class
각 클래스에 코드가 중복된다.
상속
Vehicle
speedUp()
speedDown()
superclass
Car
Truck
Bus
subclass

상속 계층도

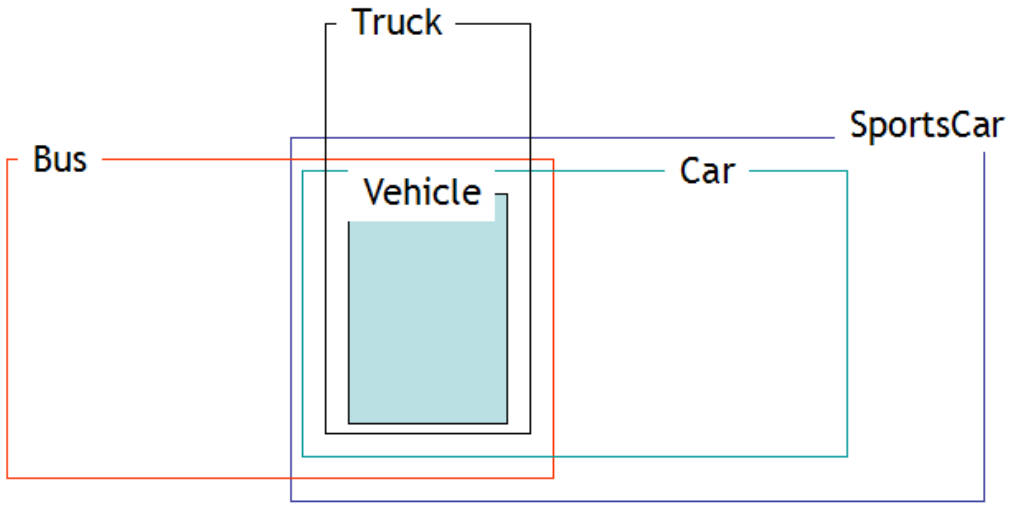
- 상속은 여러 단계로 이루어질 수 있다.



상속과 클래스 멤버




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



is-a 관계

- 상속은 is-a 관계 표현
  - 자동차는 탈것이다. (Car *is a* Vehicle).
  - 사자, 개, 고양이는 동물이다. (Lion *is a* Animal)

- has-a(포함) 관계
  - 도서관은 책을 가지고 있다(Library *has a* book).
  - 거실은 소파를 가지고 있다.

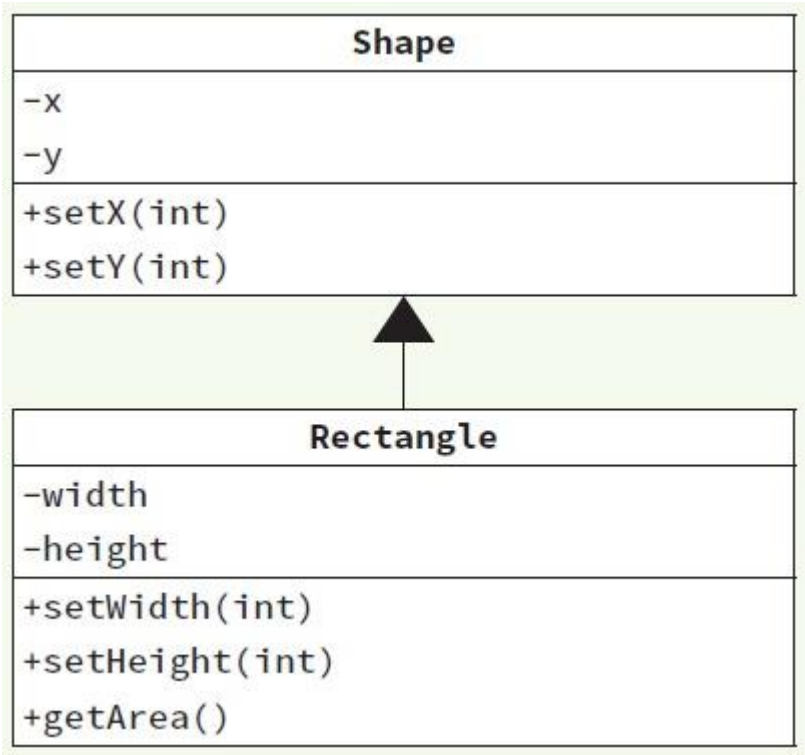
상속 예제

부모 클래스	자식 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)

Lab: 도형

- 사각형(Rectangle)과 원(Circle)의 도형 객체를 고려한다. 각 객체에 대한 클래스를 정의하라.
  - 사각형은 좌표, 너비(width), 폭(height)으로 정의된다.
  - 원은 좌표, 반지름(radius)으로 정의된다.

Lab: 도형 (2)


Shape
-x
-y
+setX(int)
+setY(int)
Rectangle
-width
-height
+setWidth(int)
+setHeight(int)
+getArea()

```
class Shape {
        int x, y;
public:
        void setX(int xval) { x = xval; }
        void setY(int yval) { y = yval; }
};

class Rectangle : public Shape {
        int width, height;
public:
        void setWidth(int w) { width = w; }
        void setHeight(int h) { height = h; }
        int getArea() { return (width * height); }
};
```

```
int main() {
  Rectangle r;

  r.setWidth(5);
  r.setHeight(6);

  cout << "사각형의 면적: " << r.getArea() << endl;

  return 0;
}
```

상속 관계의 생성자와 소멸자 실행

- 자식 클래스의 객체가 생성될 때 자식 클래스의 생성자와 기본 클래스의 생성자가 모두 실행되는가? 아니면 자식 클래스의 생성자만 실행되는가?
- 자식 클래스의 생성자와 기본 클래스의 생성자 중에서 어떤 생성자가 먼저 실행되는가?
- 객체가 소멸될 경우에 그 순서는?

상속에서의 생성자와 소멸자

- 부모 클래스의 생성자가 먼저 실행된 후에 자식 클래스의 생성자가 실행
- 자식 클래스의 소멸자가 먼저 실행되고, 부모 클래스의 소멸자가 나중에 실행

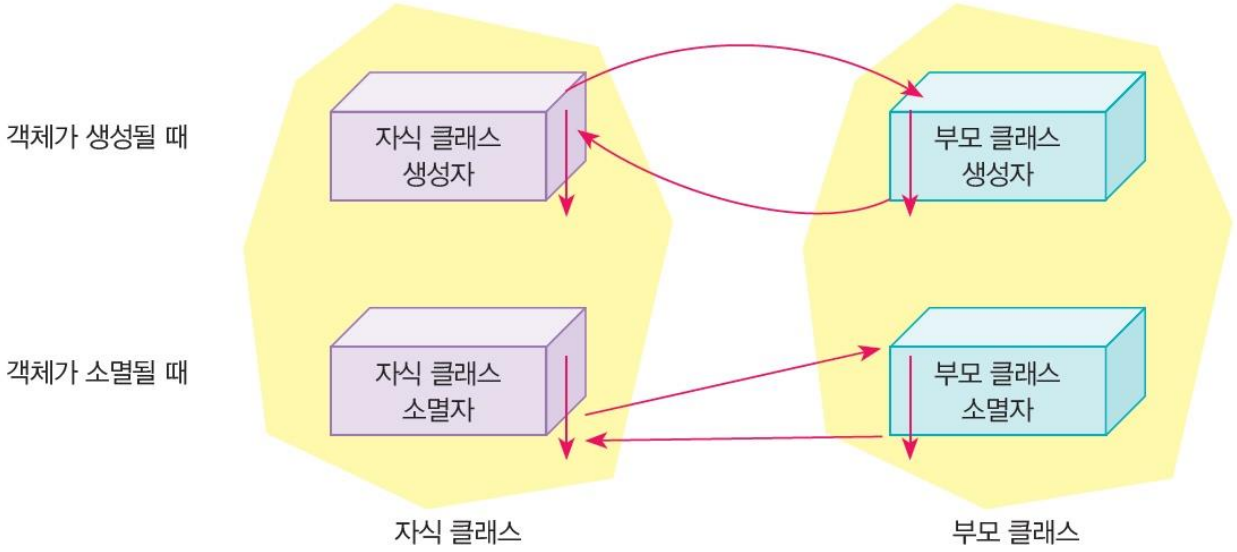
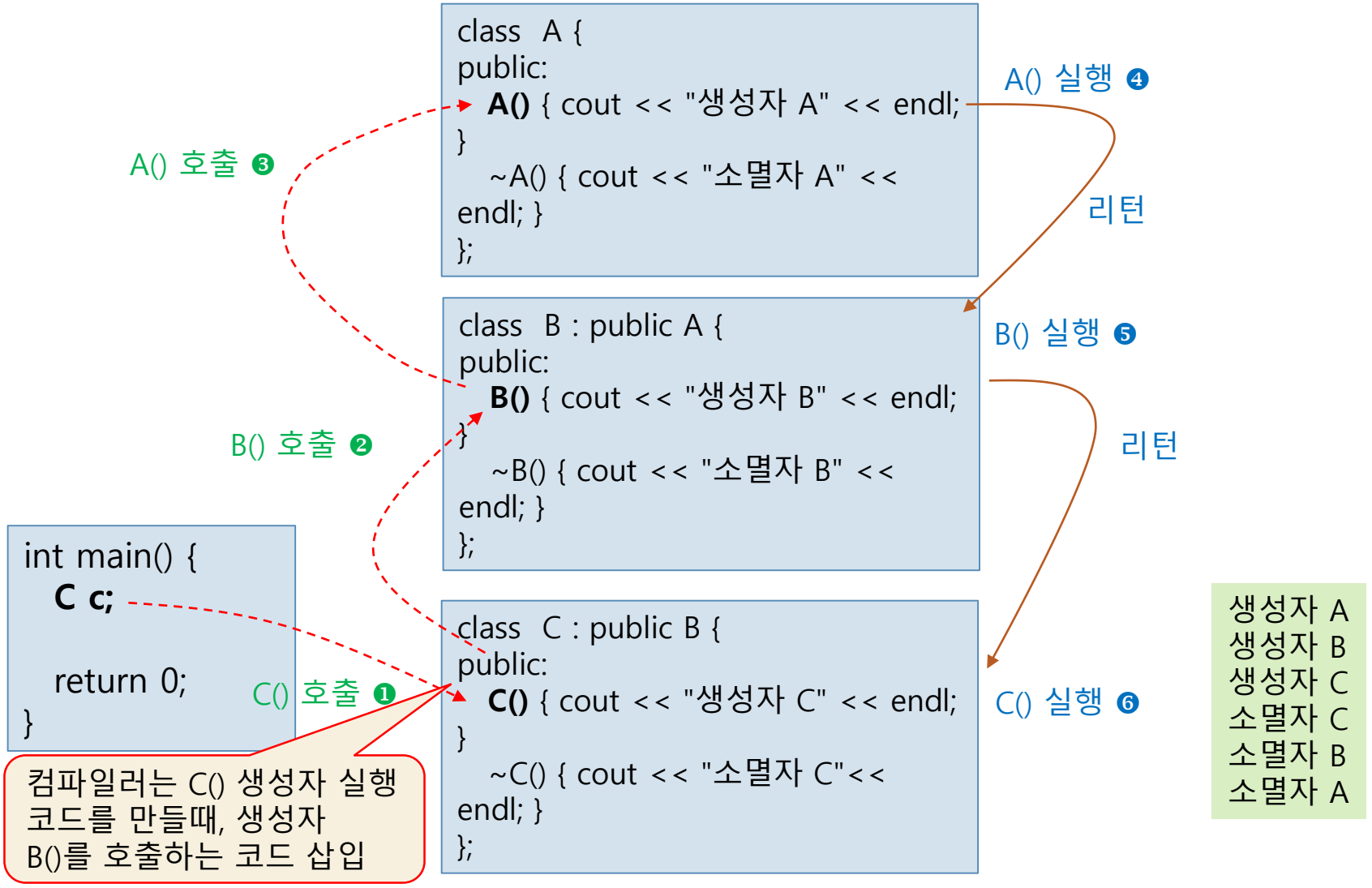


그림 11.8 상속에서 생성자와 소멸자의 호출

생성자 호출 관계 및 실행 순서



```
class Shape {
        int x, y;
public:
        Shape() {                   cout << "Shape 생성자() " << endl; }
        ~Shape() {                  cout << "Shape 소멸자() " << endl; }
};

class Rectangle : public Shape {
        int width, height;
public:
        Rectangle() {               cout << "Rectangle 생성자()" << endl;
        }
        ~Rectangle() {              cout << "Rectangle 소멸자()" << endl;
        }
};
```

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

부모 클래스 생성자 호출

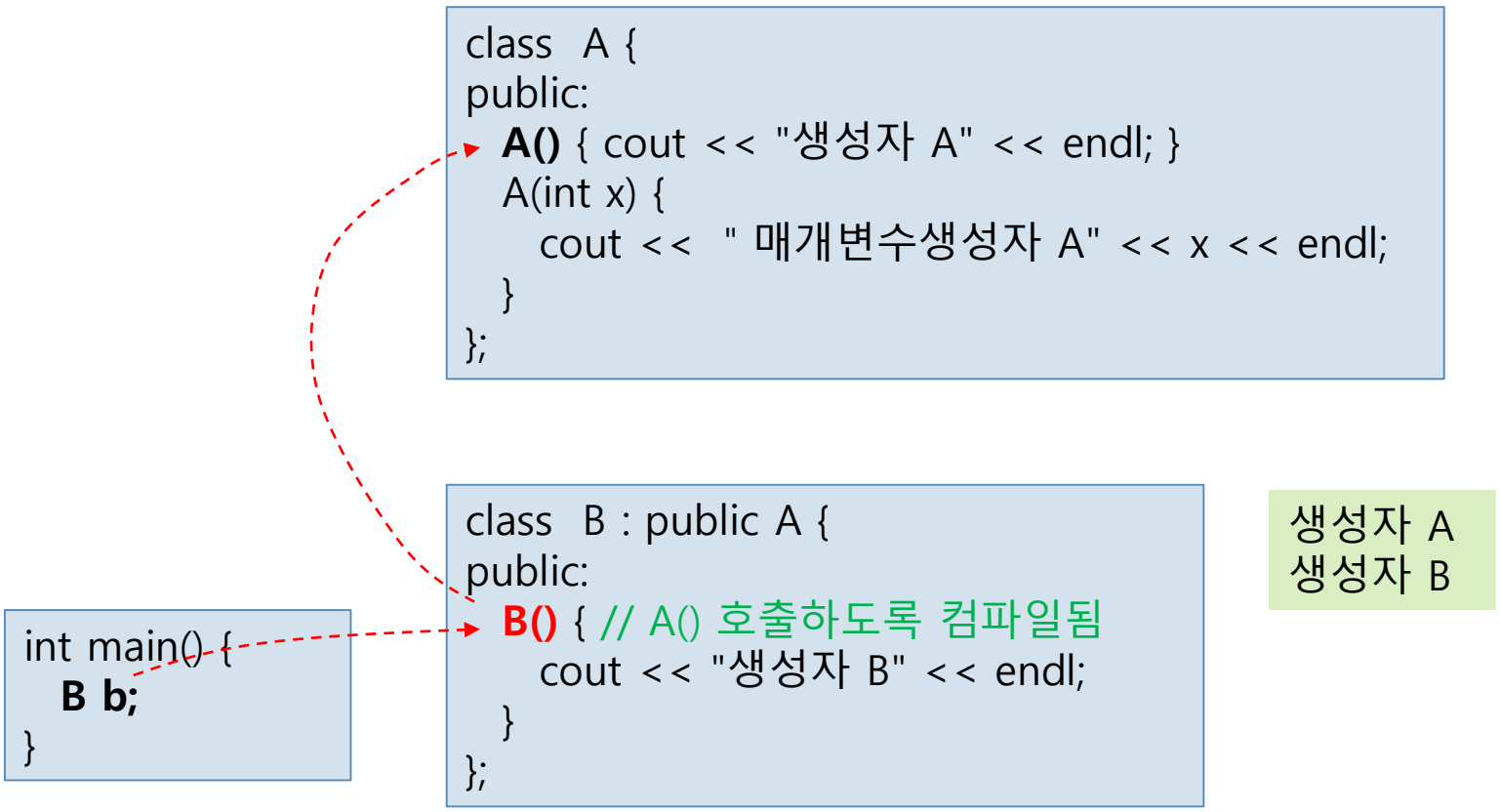
- 부모 클래스의 생성자를 명시하지 않으면, 항상 기본 생성자가 호출
- 다른 부모 생성자를 호출하려면 해당 부모 생성자를 다음과 같이 명시

문법 11.2 자식 클래스의 생성자 호출

```
자식클래스의 생성자() : 부모클래스의 생성자()
{
}
```

부모클래스 생성자가 명시되지 않으면?

- 컴파일러는 기본 생성자를 호출



부모클래스에 기본 생성자가 없으면?

```
class   A {
public:

   A(int x) {
      cout <<   " 매개변수생성자 A" << x << endl;
   }
};
```

```
class   B : public A {
public:
   B() { // A() 호출하도록 컴파일됨
      cout << "생성자 B" << endl;
   }
};
```

```
int main() {
   B b;
}
```

자식클래스 생성자가 매개 변수를 갖는 경우는?

- 부모클래스의 기본 생성자 호출

```
class  A {
public:
  A() { cout << "생성자 A" << endl; }
  A(int x) {
    cout <<  " 매개변수생성자 A" << x << endl;
  }
};
```

```
class  B : public A {
public:
  B() { // A() 호출하도록 컴파일됨
    cout << "생성자 B" << endl;
  }
  B(int x) { // A() 호출하도록 컴파일됨
    cout << "매개변수생성자 B" << x << endl;
  }
};
```

```
int main() {
  B b(5);
}
```

자식 클래스 생성자에서 부모 클래스의 생성자 선택

```
class  A {
public:
   A() {  cout << "생성자 A" << endl; }
   A(int x) {
      cout << "매개변수생성자 A" << x << endl;
   }
};
```

```
class  B : public A {
public:
   B() { // A() 호출하도록 컴파일됨
      cout << "생성자 B" << endl;
   }
   B(int x) :  A(x+3) {
      cout << "매개변수생성자 B" << x << endl;
   }
};
```

```
int main() {
   B b(5);
}
```

컴파일러의 기본 생성자 호출 코드 삽입

컴파일러가 묵시적으로 삽입한 코드

```
class B {
  B() : A() {
    cout << "생성자 B" << endl;
  }

  B(int x) : A() {
    cout << "매개변수생성자 B" << x << endl;
  }
};
```

부모 클래스 생성자 호출

- 부모 클래스 생성자 호출 다음에 멤버 초기화 리스트 작성 가능

```
Rectangle(int x=0, int y=0, int w=0, int h=0) : Shape(x, y)
{
   width = w;
   height = h;
}
```

```
Rectangle(int x=0, int y=0, int w=0, int h=0) : Shape(x, y), width(w), height(h)
{}
```

```
class Shape {
        int x, y;
public:
        Shape() {
                cout << "Shape 생성자() " << endl;
        }
        Shape(int xloc, int yloc) : x{xloc}, y{yloc} {
                cout << "Shape 생성자(xloc, yloc) " << endl;
        }
        ~Shape() {
                cout << "Shape 소멸자() " << endl;
        }
};
```

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

```
int main()
{
  Rectangle r(0, 0, 100, 100);
  return 0;
}
```

Lab: 컬러 사각형

- 사각형을 Rect 클래스로 나타내자. 이 클래스를 상속받아서 컬러 사각형 ColoredRect을 정의하라. ColoredRect 클래스를 이용하여 화면에 다음과 같은 색깔있는 사각형을 그려보자.

```
class Rect {
protected:
        int x, y, width, height;
public:
        Rect(int x, int y, int w, int h) : x(x), y(y), width(h), height(h) { }
        void draw()
        {
                HDC hdc = GetWindowDC(GetForegroundWindow());
                Rectangle(hdc, x, y, x + width, y + height);
        }
};
```

```
class ColoredRect : Rect {
        int red, green, blue;
public:
        ColoredRect(int x, int y, int w, int h, int r, int g, int b) :
                Rect(x, y, h, w), red(r), green(g), blue(b) { }
        void draw()
        {
                HDC hdc = GetWindowDC(GetForegroundWindow());
                SelectObject(hdc, GetStockObject(DC_BRUSH));
                SetDCBrushColor(hdc, RGB(red, green, blue));
                Rectangle(hdc, x, y, x + width, y + height);
        }
};
```

예제

```
int main()
{
        Rect r1(100, 100, 80, 80);
        ColoredRect r2(200, 200, 80, 80, 255, 0, 0);

        r1.draw();
        r2.draw();
        return 0;
}
```

접근 지정자에 따른 상속 기준

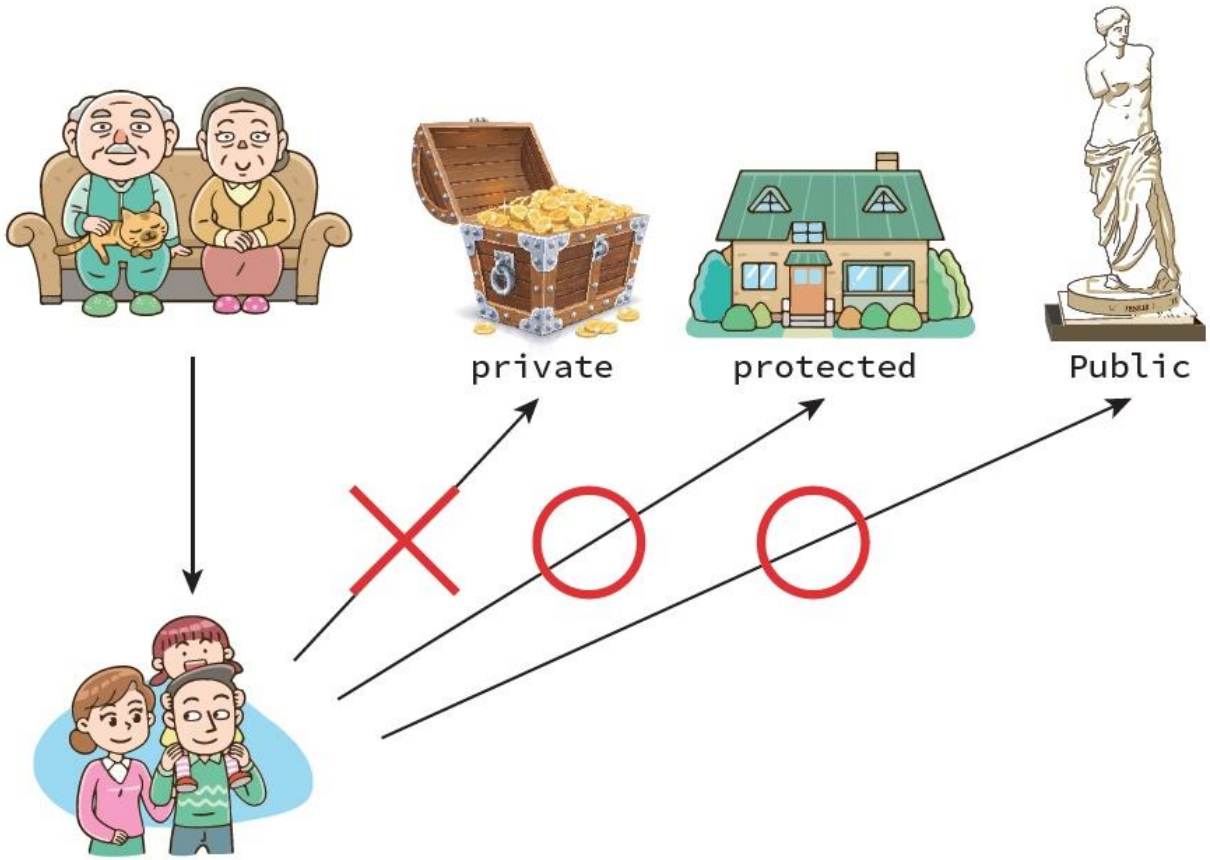
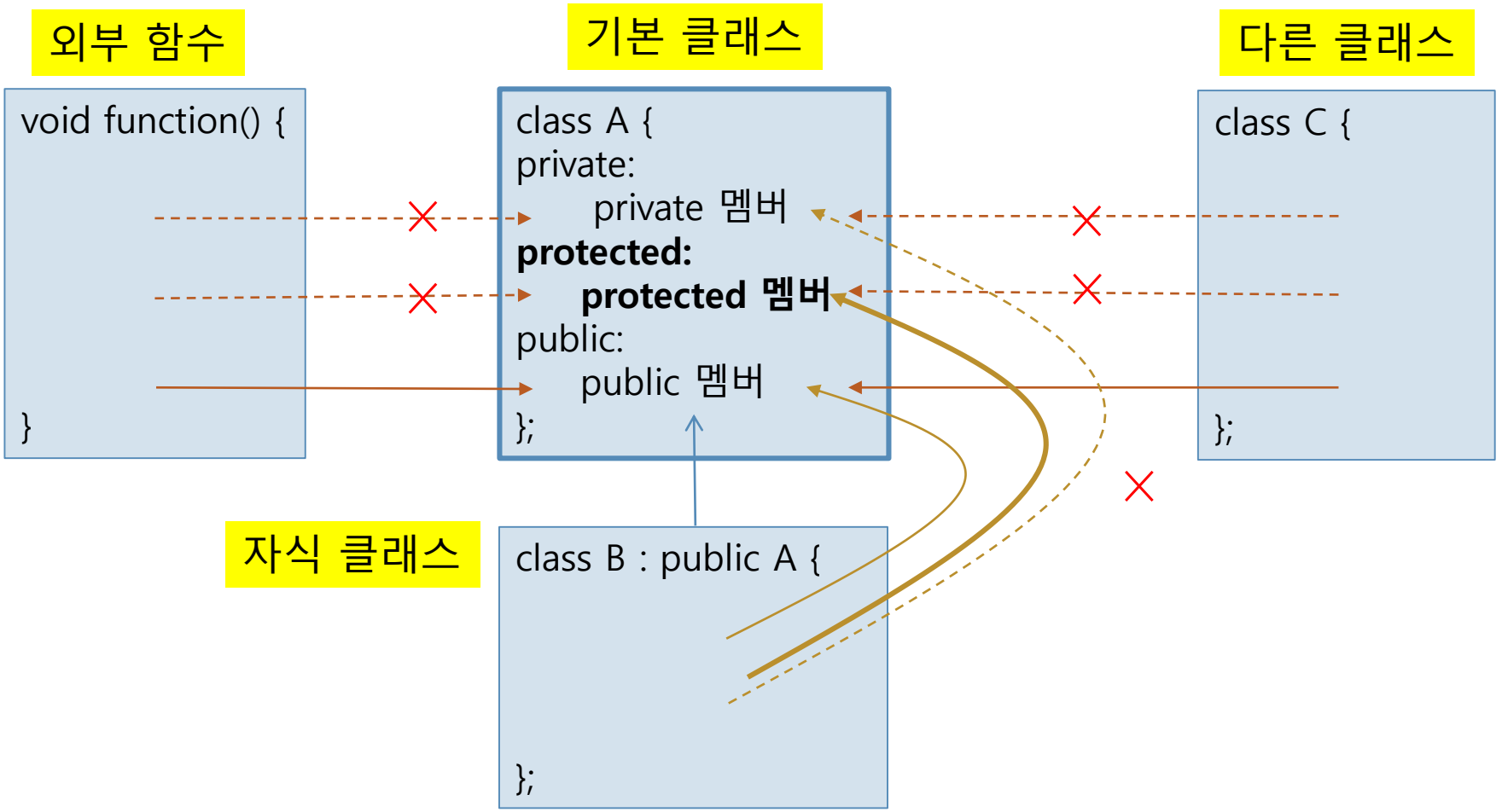


그림 11.9 상속에서의 접근 지정자

접근 지정자에 따른 상속 기준(2)

- private 멤버
  - 선언된 클래스 내에서만 접근 가능
  - 자식 클래스에서도 기본 클래스의 private 멤버 직접 접근 불가
- Public 멤버
  - 선언된 클래스나 외부 어떤 클래스, 외부 함수에 접근 허용
  - 자식 클래스에서 기본 클래스의 public 멤버 접근 가능
- Protected 멤버
  - 선언된 클래스에서 접근 가능
  - 자식 클래스에서 접근 허용
  - 다른 클래스나 외부 함수에서는 접근 불허

접근 지정에 따른 접근성


외부 함수
void function() {
}
기본 클래스
class A {
private:
private 멤버
protected:
protected 멤버
public:
public 멤버
};
다른 클래스
class C {
};
자식 클래스
class B : public A {
};

```
class Person {
        string name;
protected:
        string address;
};

class Student : public Person {
public:
        void setAddress(string add) {
                address = add;
        }
        string getAddress() {
                return address;
        }
};
```

```
int main() {
        Student obj;

        obj.setAddress("서울시 종로구 1번지");
        cout << obj.getAddress() << endl;

        return 0;
}
```

멤버 함수 재정의

- 자식 클래스가 필요에 따라 상속된 멤버 함수를 재정의하여 사용하는 것을 의미
- 함수 재정의를 오버라이딩(overriding)이라 함

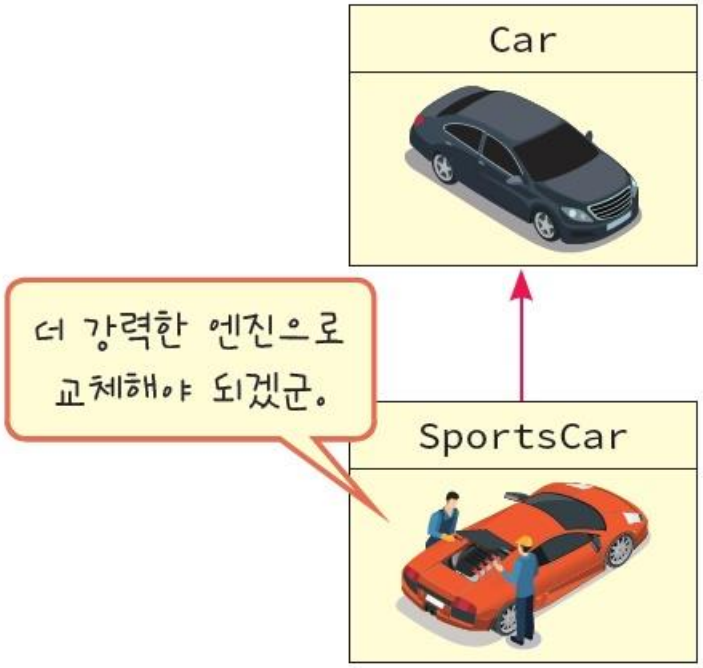


그림 11.10 멤버 함수 재정의

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

class Animal {
public:
        void speak()
        {
                cout << "동물이 소리를 내고 있음" << endl;
        }
};
class Dog : public Animal {
public:

        void speak()
        {
                cout << "멍멍!" << endl;
        }
};
```

```
int main()
{
        Dog obj;
        obj.speak();
        return 0;
}
```

함수의 중복과 재정의

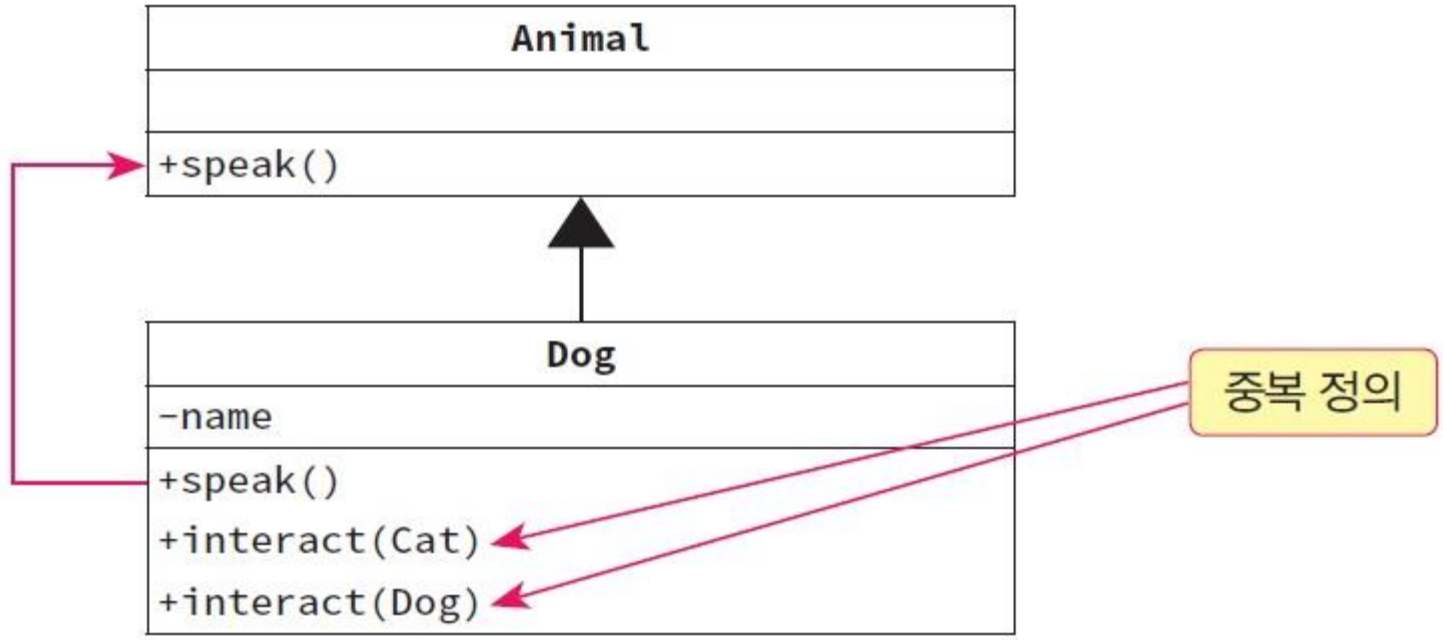


그림 11.11 재정의와 중복정의의 비교

부모 클래스의 멤버 함수 호출

- 상속에서 부모 클래스의 재정의된 멤버 함수를 호출하려면 '::'을 사용하여 부모클래스를 명시하면 됨
  - ParentClass::print()
- 보통, 자식클래스의 멤버 함수는 부모 클래스 버전을 먼저(혹은 나중에) 호출하여 일정 작업을 한 후에 자신의 역할을 수행하는 방식으로 작성됨

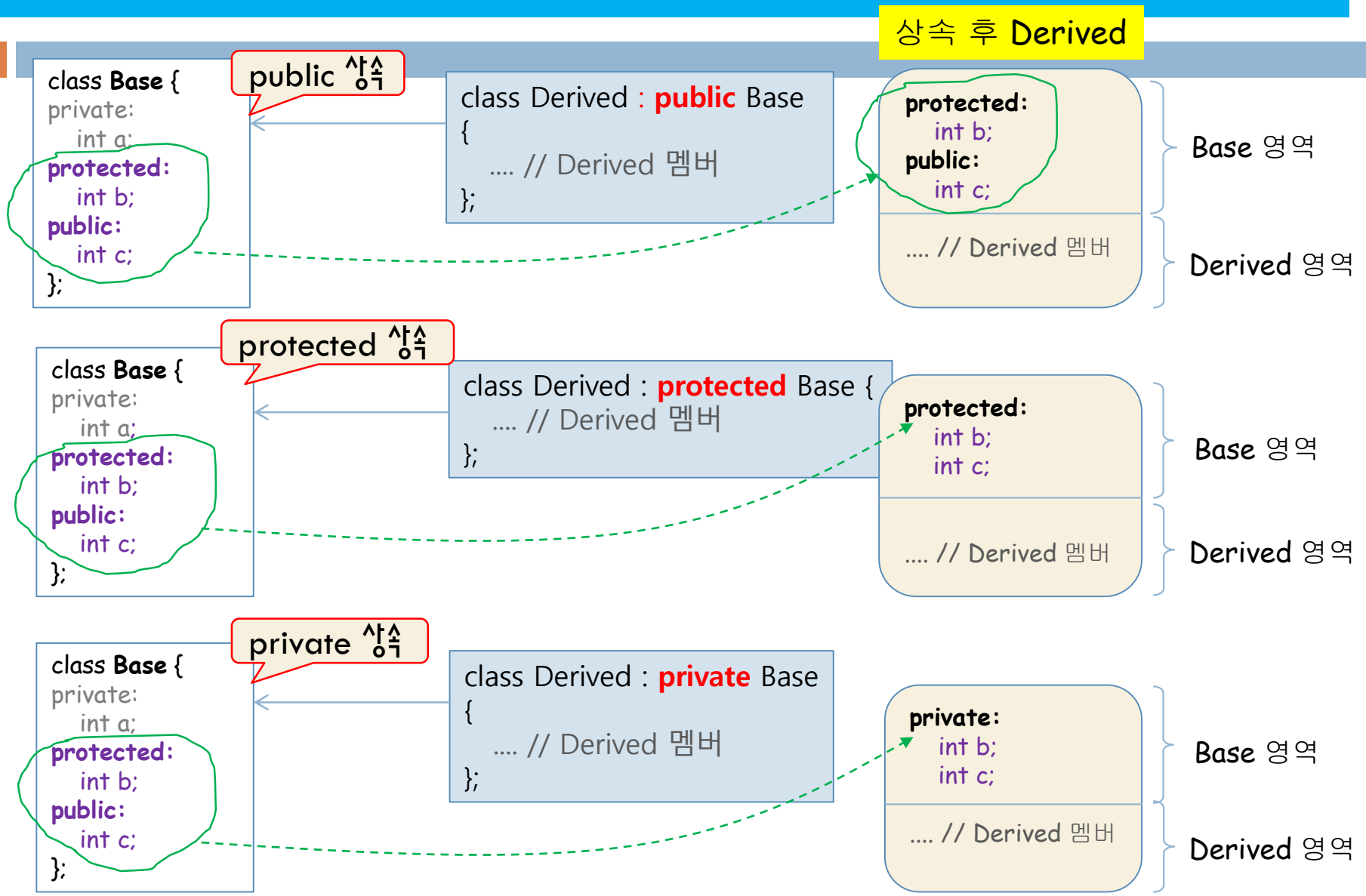
예제

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

class ParentClass {
public:
        void print() {
                cout << "부모 클래스의 print() 멤버 함수" << endl;
        }
};

class ChildClass : public ParentClass {
        int data;
public:
        void print() { //멤버 함수 재정의
                ParentClass::print();
                cout << "자식 클래스의 print() 멤버 함수 " << endl;
        }
};
```

상속 접근 지정자

- 상속 선언 시 public, private, protected의 3가지 중 하나 지정
- 기본 클래스의 멤버의 접근 속성을 어떻게 계승할지 지정
  - public – 기본 클래스의 protected, public 멤버 속성을 그대로 계승
  - private – 기본 클래스의 protected, public 멤버를 private으로 계승
  - protected – 기본 클래스의 protected, public 멤버를 protected로 계승
- 상속 접근지정자 생략시에는 private 상속으로 처리됨에 유의할 것

상속 시 접근 지정에 따른 멤버의 접근 지정 속성 변화

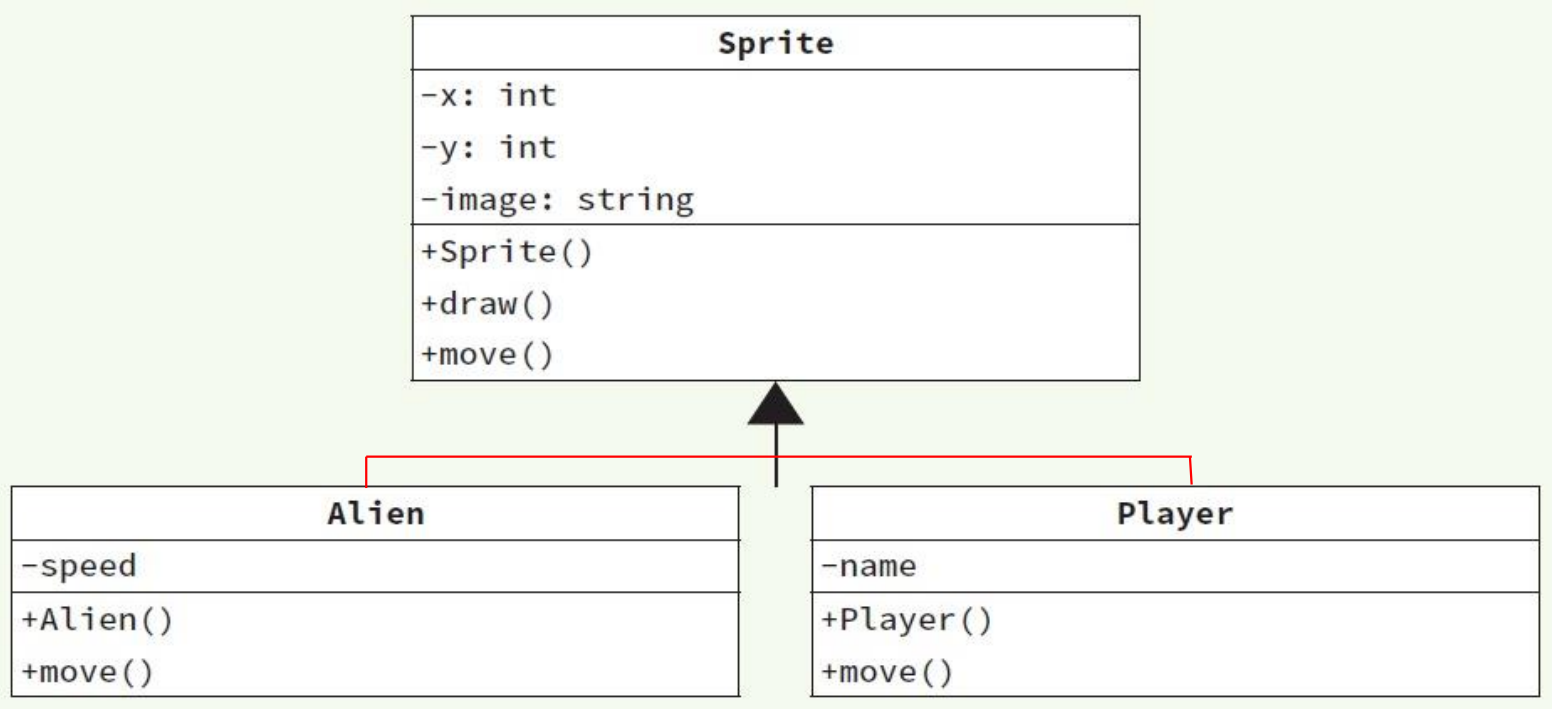


예제

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

class Base {
        public: int x;
        protected: int y;
        private: int z;
};

class Derived : private Base{

};

int main() {
        Derived obj;
        cout << obj.x; // ?
}
```

Lab: 게임에서의 상속 모델링

- Alien, Player는 미사일, 경기자를 표현
- 이들은 화면에서 움직이는 작은 그림인 sprite의 공통부분을 가짐



예제

```
int main()
{
   Alien a( 0, 100, "image1.jpg" );
   Player p(0, 100, "image1.jpg");
   return 0;
}
```

```
class Sprite {
        int x, y;
        string image;
public:
        Sprite(int x, int y, string image) : x(x), y(y), image(image) { }
        void draw() { }
        void move() { }
};
class Alien : public Sprite {
        int speed;
public:
        Alien(int x, int y, string image) : Sprite(x, y, image) { }
        void move() { }
};

class Player : public Sprite {
        string name;
public:
        Player(int x, int y, string image) : Sprite(x, y, image) { }
        void move() { }
};
```

다중 상속

- 다중 상속(multiple inheritance)이란 하나의 자식 클래스가 두개 이상의 부모 클래스로부터 멤버를 상속받는 것