객체지향언어

제8장 상속

2025. 4. 5

컴퓨터공학

허훈식

Learning Objectives

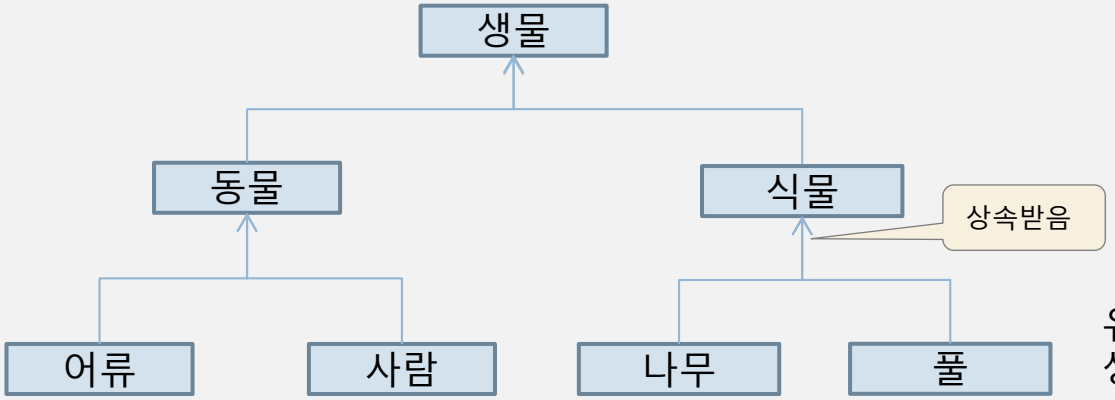
1. C++ 객체 지향 상속의 개념을 이해한다.
2. 상속을 선언하는 방법을 알고, 파생 클래스의 객체에 대해 이해한다.
3. 업 캐스팅과 다운 캐스팅 등 상속과 객체 포인터 사이의 관계를 이해한다.
4. protected 접근 지정에 대해 이해한다.
5. 상속 관계에 있는 파생 클래스의 생성 및 소멸 과정을 이해한다.
6. public, protected, private 상속의 차이점을 이해한다.
7. 다중 상속을 선언하고 활용할 수 있다.
8. 다중 상속을 문제점을 이해하고, 가상 상속으로 해결할 수 있다.

8.1 상속의 개념

➢ 유전적 상속과 객체 지향 상속

유산 상속

유전적 상속 : 객체 지향 상속



유전적 상속과 관계된 생물 분류

8.1 상속의 개념

- C++에서의 상속(Inheritance)
  - 기존 클래스를 재사용(확장)하여 새로운 클래스를 작성하는 것.
  - 기본 클래스의 속성과 기능을 파생 클래스에 물려주는 것

- 상속 관계에서의 클래스
  - 기본 클래스(base class) - 상속해주는 클래스. 부모 클래스
  - 파생 클래스(derived class) - 상속받는 클래스. 자식 클래스
    - 기본 클래스의 속성과 기능을 물려받고 자신 만의 속성과 기능을 추가하여 작성
  - 상속 되는 것 : 멤버 변수, 멤버 함수

8.1 상속의 개념

➢ 상속의 표현

8.1 상속의 개념

➢ 상속의 목적 및 장점

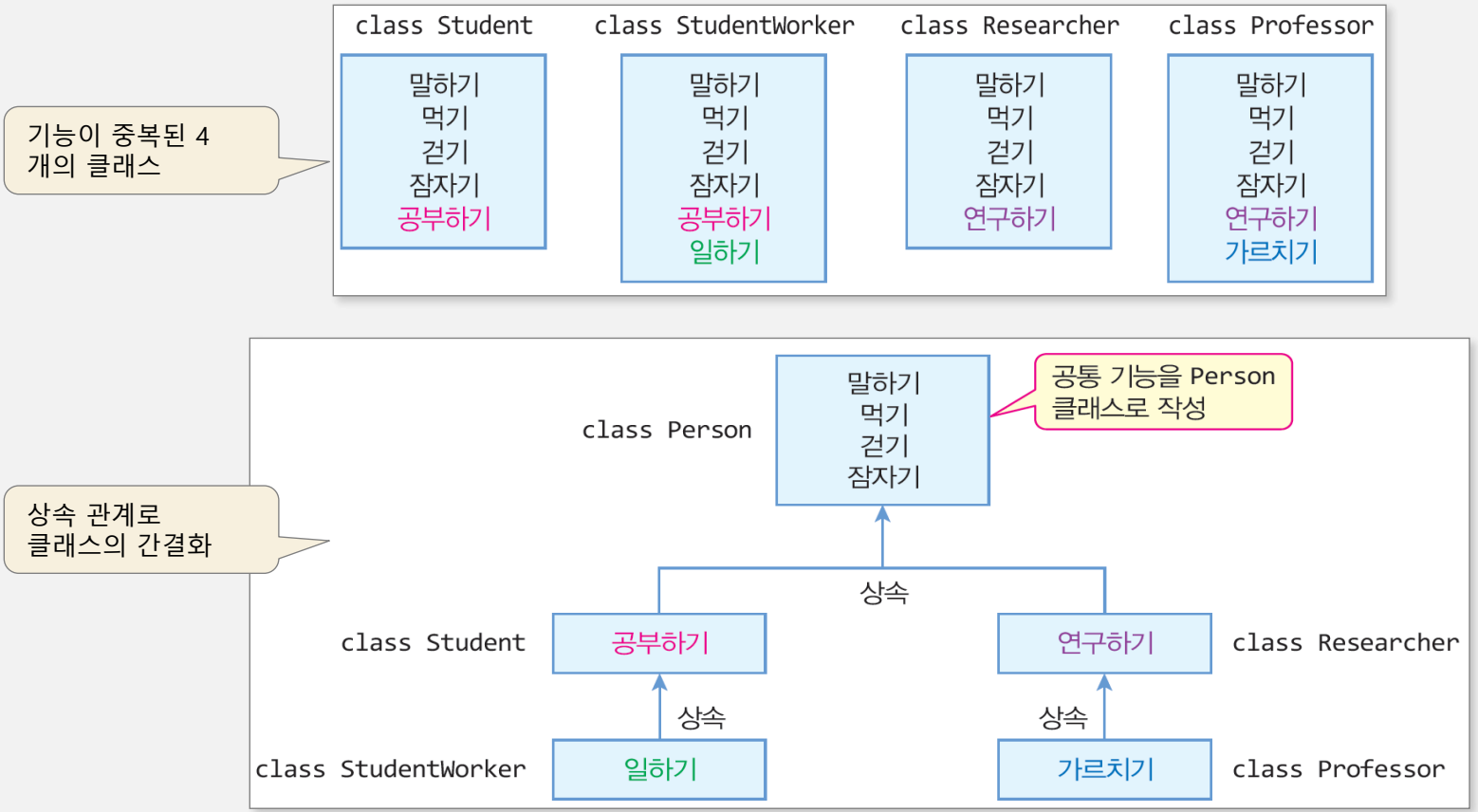
- 간결한 클래스 작성
  - 기본 클래스의 기능을 물려받아 파생 클래스를 간결하게 작성

- 클래스 간의 계층적 분류 및 관리의 용이함
  - 상속은 클래스들의 구조적 관계 파악 용이

- 클래스 재사용과 확장을 통한 소프트웨어 생산성 향상
  - 빠른 소프트웨어 생산 필요
  - 기존에 작성한 클래스의 재사용 – 상속 (새로운 기능을 확장)
  - 앞으로 있을 상속에 대비한 클래스의 객체 지향적 설계 필요

8.1 상속의 개념

➢ 상속 관계로 클래스의 간결화 사례



8.1 상속의 개념

➢ 상속을 위한 기본 조건인 is-a 관계

- 상속 특징
  - 파생 클래스는 기본 클래스 특성과 함께 자신만의 추가 특성을 갖는다.

A is a B (A는 일종의 B이다.)
두 클래스간에 상속 관계가 성립하려면 is-a 관계가 성립해야 한다.

노트북 컴퓨터는 컴퓨터이다.	class Notebook : public computer
무선전화기는 전화기이다.	class MobilePhone : public Phone
호랑이는 동물이다.	class Tiger : public Animal
한국은 국가이다.	class Korea : public nation

8.1 상속의 개념

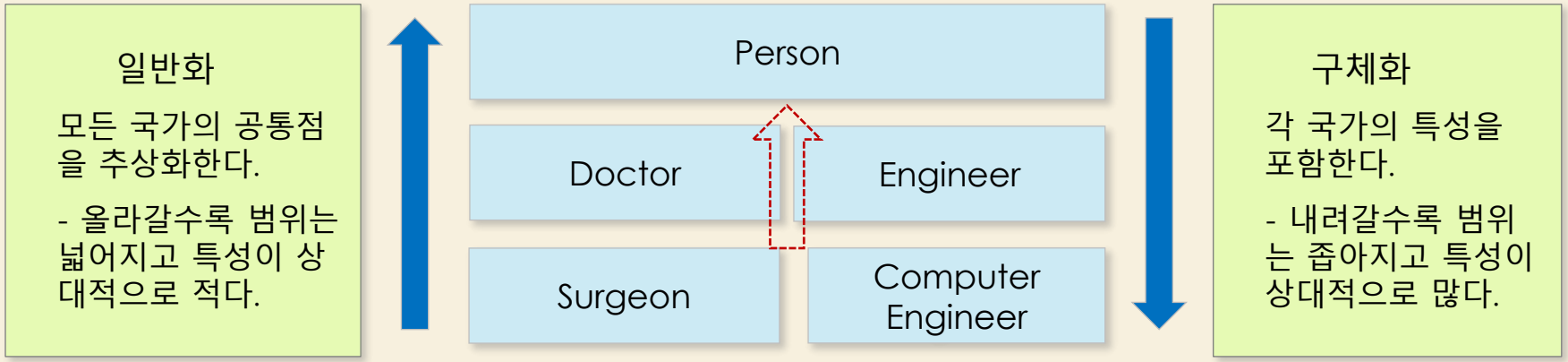
➢ 왜 is-a 관계가 성립되어야 하는가?

- 아래로 내려갈수록 범위가 좁아지고 특성이 많다는 의미는 Derived 클래스 내 멤버가 많아진다. 이는 C++의 상속 특징과 일치한다.

-> is-a 관계 성립이 상속 조건에 반드시 필요.

❖ A is a B (A는 일종의 B이다.)

- 외과의사는 사람이다



8.2 클래스 상속과 객체

➢ 상속 선언

- Student 클래스는 Person 클래스의 멤버를 물려받는다.
- StudentWorker 클래스는 Student의 멤버를 물려받는다.
  - Student가 물려받은 Person의 멤버도 함께 물려받는다.

```
class Student : public Person {
        // Person을 상속받는 Student 선언
        .....
};

class StudentWorker : public Student {
        // Student를 상속받는 StudentWorker 선언
        .....
};
```

8.2 클래스 상속과 객체

➢ [예제 8-1] Point 클래스를 상속받는 ColorPoint 클래스 만들기

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

// 2차원 평면에서 한점을 표현하는 클래스 Point 선언
class Point {
    int x, y;   //한 점 (x,y) 좌표값
public:
    void set(int x, int y) {
                this->x = x; this->y = y;
    }
    void showPoint() {
                cout << "(" << x << "," << y << ")"
                << endl;
    }
};
```

```
Red:(3,4)
```

```
class ColorPoint : public Point {
// 2차원 평면에서 컬러 점을 표현하는 클래스
                ColorPoint. Point를 상속받음
    string color;              // 점의 색 표현
public:
    void setColor(string color) { this->color = color; }
    void showColorPoint();
};

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

int main() {
    Point p; // 기본 클래스의 객체 생성
    ColorPoint cp; // 파생 클래스의 객체 생성
    cp.set(3,4); // 기본 클래스의 멤버 호출
    cp.setColor("Red"); // 파생 클래스의 멤버 호출
    cp.showColorPoint(); // 파생 클래스의 멤버 호출
}
```

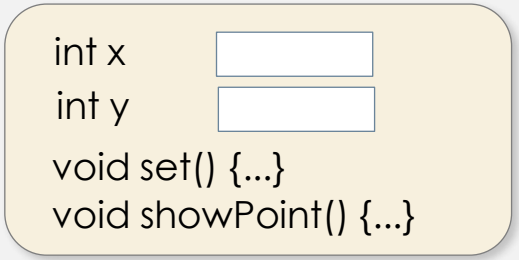
8.2 클래스 상속과 객체

➢ 파생 클래스의 객체 구성

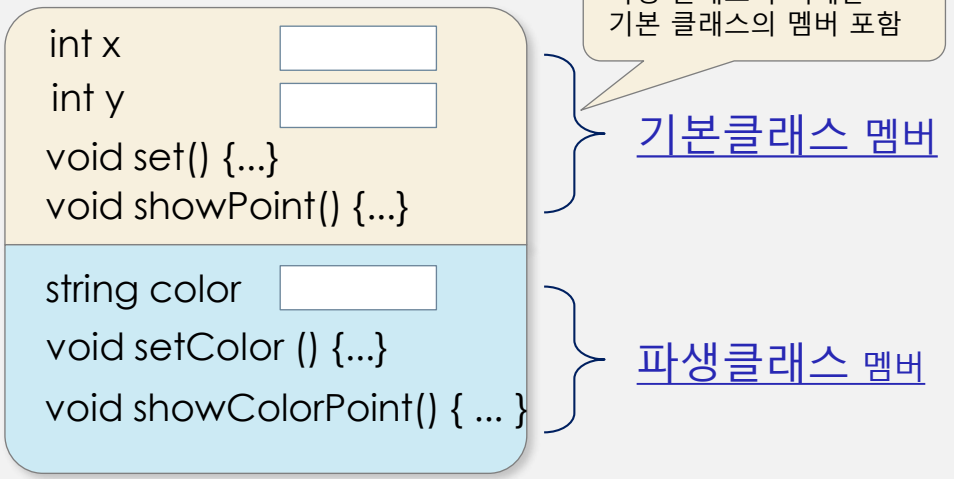
```
class Point {
        int x, y; // 한 점 (x,y) 좌표 값
public:
        void set(int x, int y);
        void showPoint();
};
```

```
class ColorPoint : public Point { // Point를 상속받음
    string color;          // 점의 색 표현
public:
    void setColor(string color);
    void showColorPoint();
};
```

Point p;

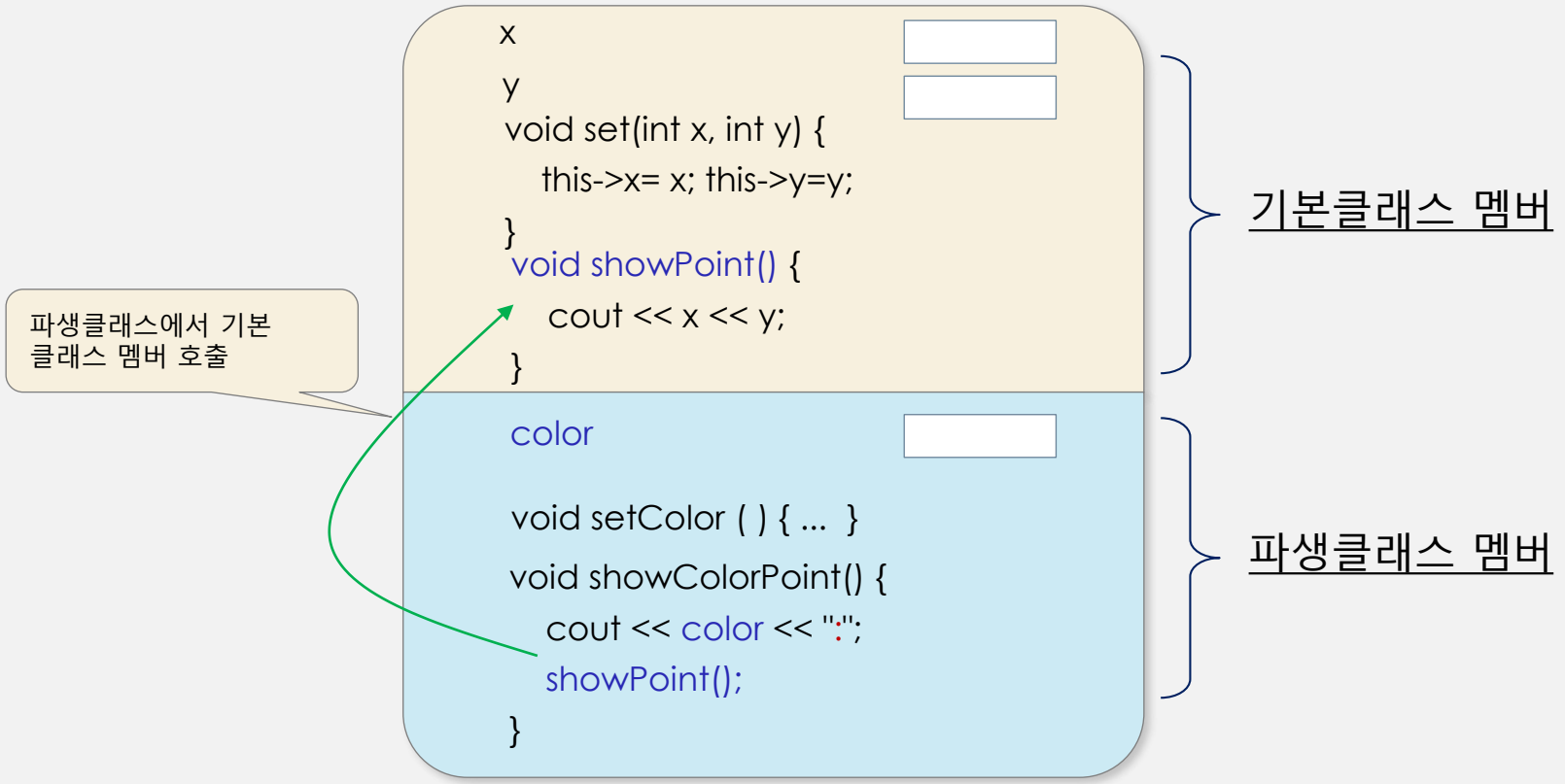


ColorPoint cp;



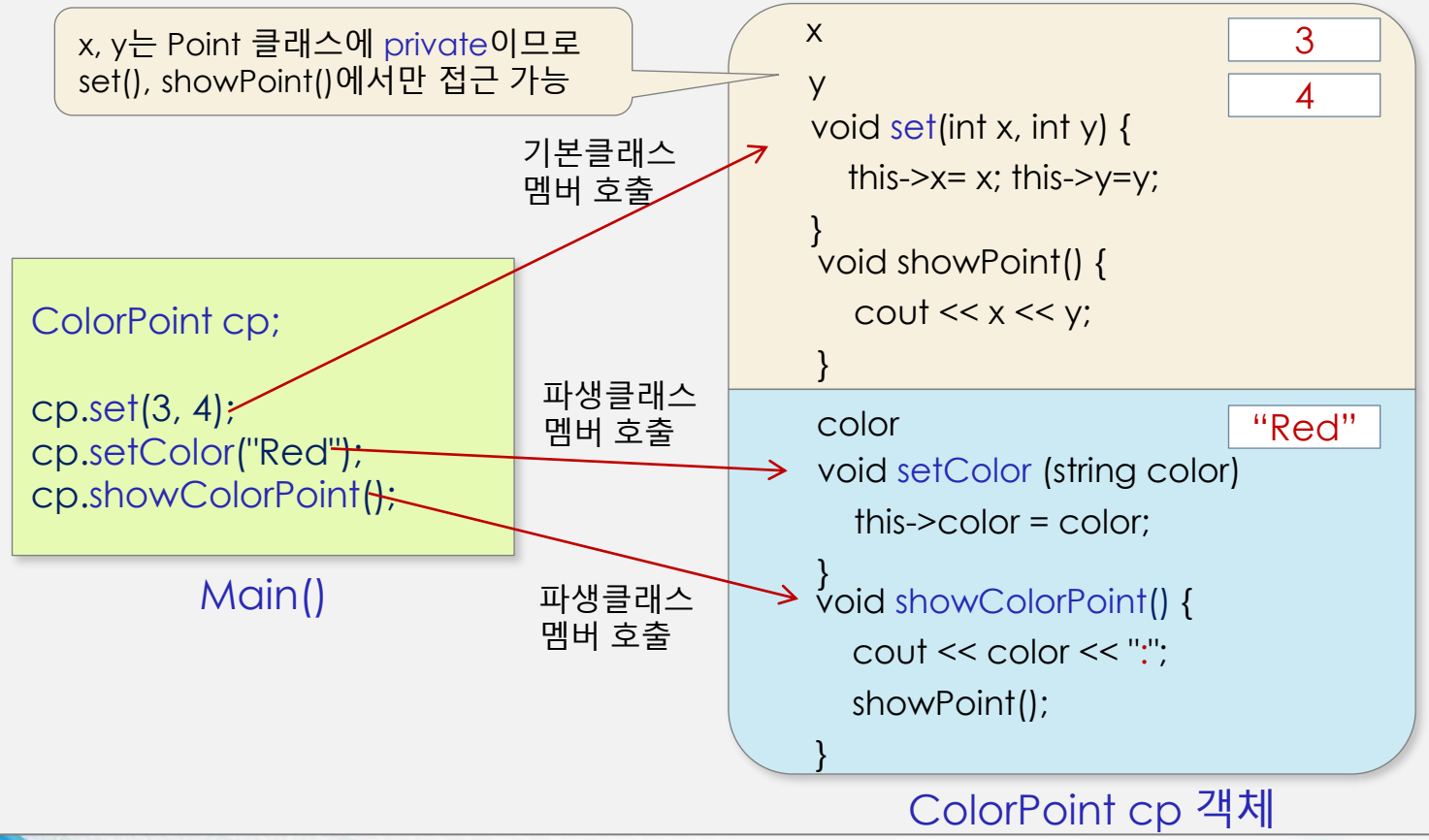
8.2 클래스 상속과 객체

➢ 파생 클래스에서 기본 클래스 멤버 접근



8.2 클래스 상속과 객체

➢ 외부에서 파생 클래스 객체에 대한 접근



8.3 상속과 객체 포인터

➢ 상속과 객체 포인터 – 업 캐스팅

- 파생 클래스 포인터가 기본 클래스 포인터에 치환되는 것.
- 기본 클래스 포인터는 기본 클래스 객체 뿐만 아니라, 기본 클래스를 직접 또는 간접적으로 상속하는 파생 클래스의 객체도 가리킬 수 있다.
- C++ 컴파일러는 포인터 연산의 가능성 여부를 판단할 때, 포인터의 자료형을 기준으로 판단하지, 실제 가리키는 객체의 자료형을 기준으로 판단하지 않는다.

```
class Person { … }

class Student : public Person { … }

class PartTimeStudent
      : public Student { … }
```

```
Person  *  ptr;
ptr     =  new  Student( );
```

```
Person  *  ptr;
ptr     = new  PartTimeStudent( );
```

```
Student *  ptr;
ptr     = new  PartTimeStudent( );
```

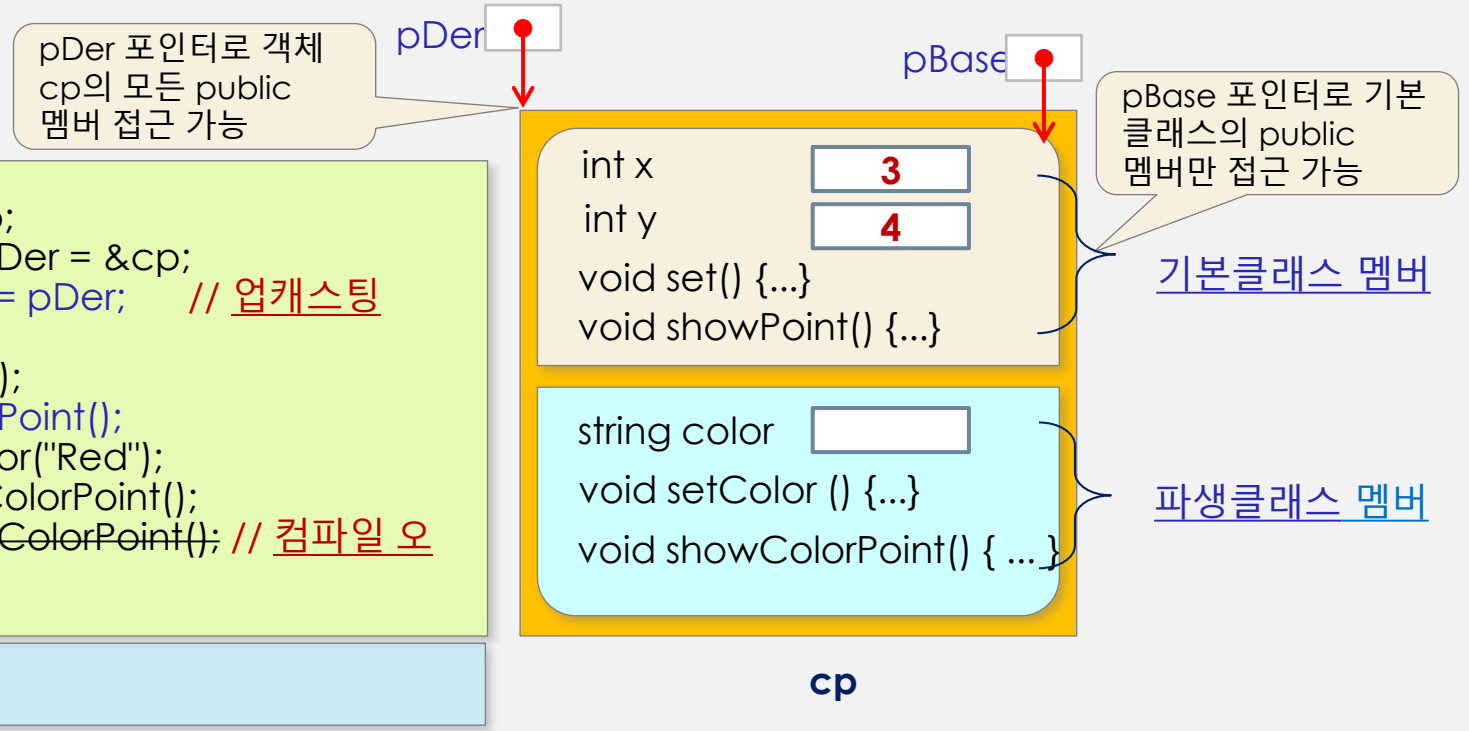
8.3 상속과 객체 포인터

➢ 업 캐스팅(up-casting)

- 업 캐스팅한 기본 클래스의 포인터는 기본 클래스의 멤버만 접근 가능하다.

```
int main() {
    ColorPoint cp;
    ColorPoint *pDer = &cp;
    Point* pBase = pDer;      // 업캐스팅

    pDer->set(3,4);
    pBase->showPoint();
    pDer->setColor("Red");
    pDer->showColorPoint();
    pBase->showColorPoint(); // 컴파일 오류
}
```

```
(3,4)
Red(3,4)
```



8.3 상속과 객체 포인터

➢ 업 캐스팅

업 캐스팅은 기본 클래스의 포인터로
파생 클래스의 객체를 가리키는 것을 말한다

업 캐스팅한 기본 클래스의 포인터는
기본 클래스의 멤버만 접근 가능하다

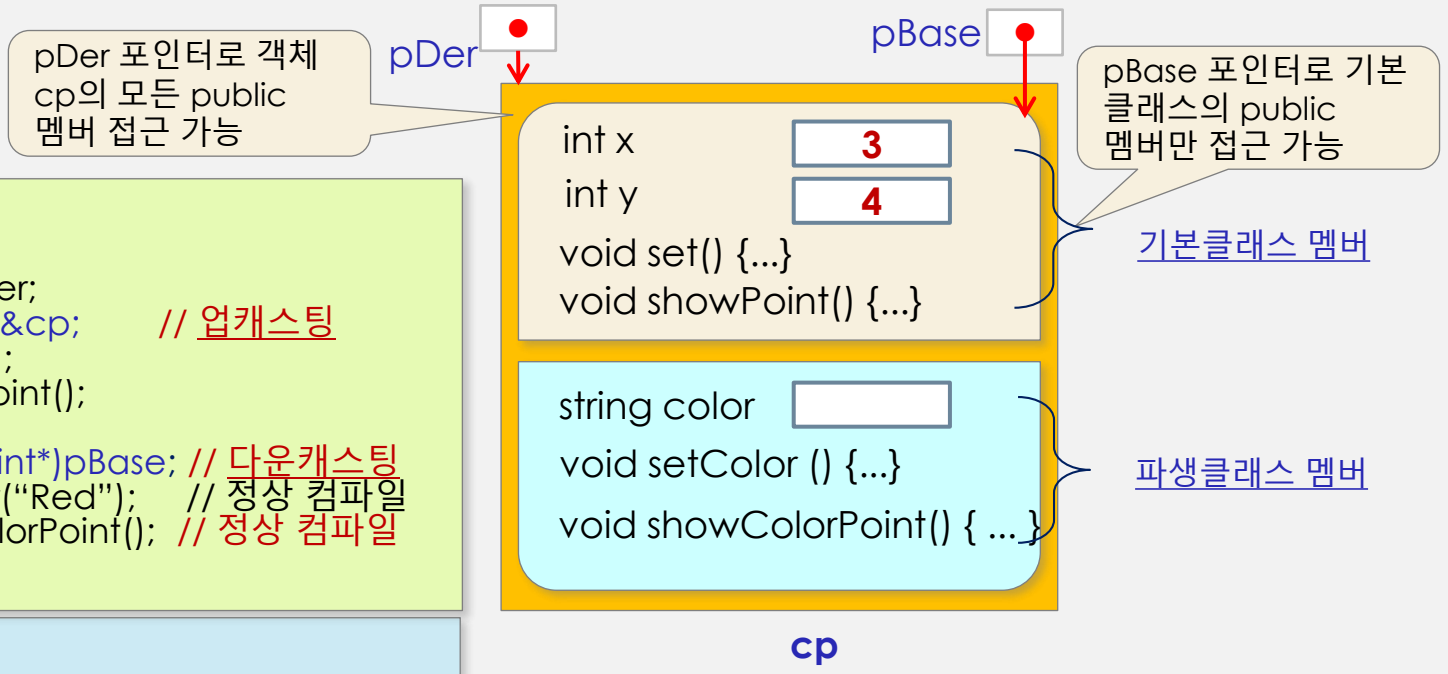
8.3 상속과 객체 포인터

➢ 상속과 객체 포인터 – 다운 캐스팅

- 기본 클래스의 포인터가 파생 클래스의 포인터에 치환되는 것
- 기본 클래스의 포인터가 가리키는 객체를 파생 클래스의 포인터로 가리킬 때는 업 캐스팅과 달리 명시적으로 타입 변환을 지정해야 한다.

```
int main() {
    ColorPoint cp;
    ColorPoint *pDer;
    Point* pBase = &cp;        // 업캐스팅
    pBase->set(3,4);
    pBase->showPoint();

    pDer=(ColorPoint*)pBase; // 다운캐스팅
    pDer->setColor("Red");     // 정상 컴파일
    pDer->showColorPoint();  // 정상 컴파일
}
```

```
(3,4)
Red(3,4)
```



8.3 상속과 객체 포인터

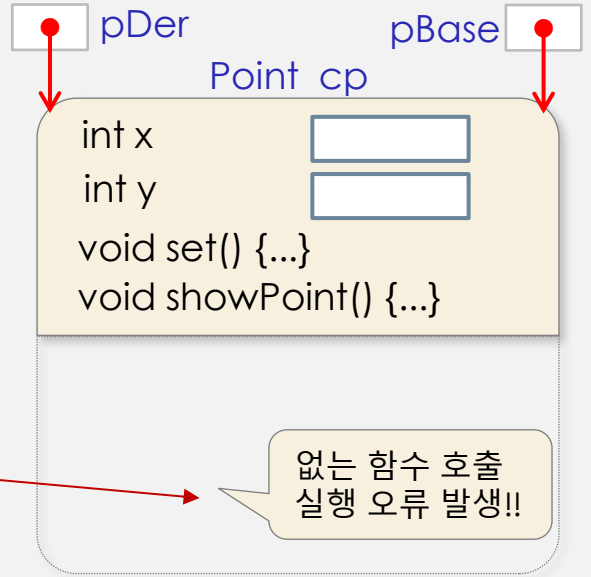
➢ 상속과 객체 포인터 – 다운 캐스팅 (잘못된 경우)

- 다운 캐스팅의 문법적 오류는 없지만 pDer이 가리키는 객체 공간에 setColor 함수가 없기 때문에 실행 중에 오류 발생

```
int main() {
    Point  po, *pBase;
    ColorPoint *pDer;

    pBase = &po;

    pDer=(ColorPoint*)pBase;

    pDer->set(3, 4);
    pDer->setColor("red");
}
```

잘못된 다운캐스팅

SetColor()는 ColorPoint의 멤버이므로 컴파일 오류는 없음



8.4 protected 접근 지정

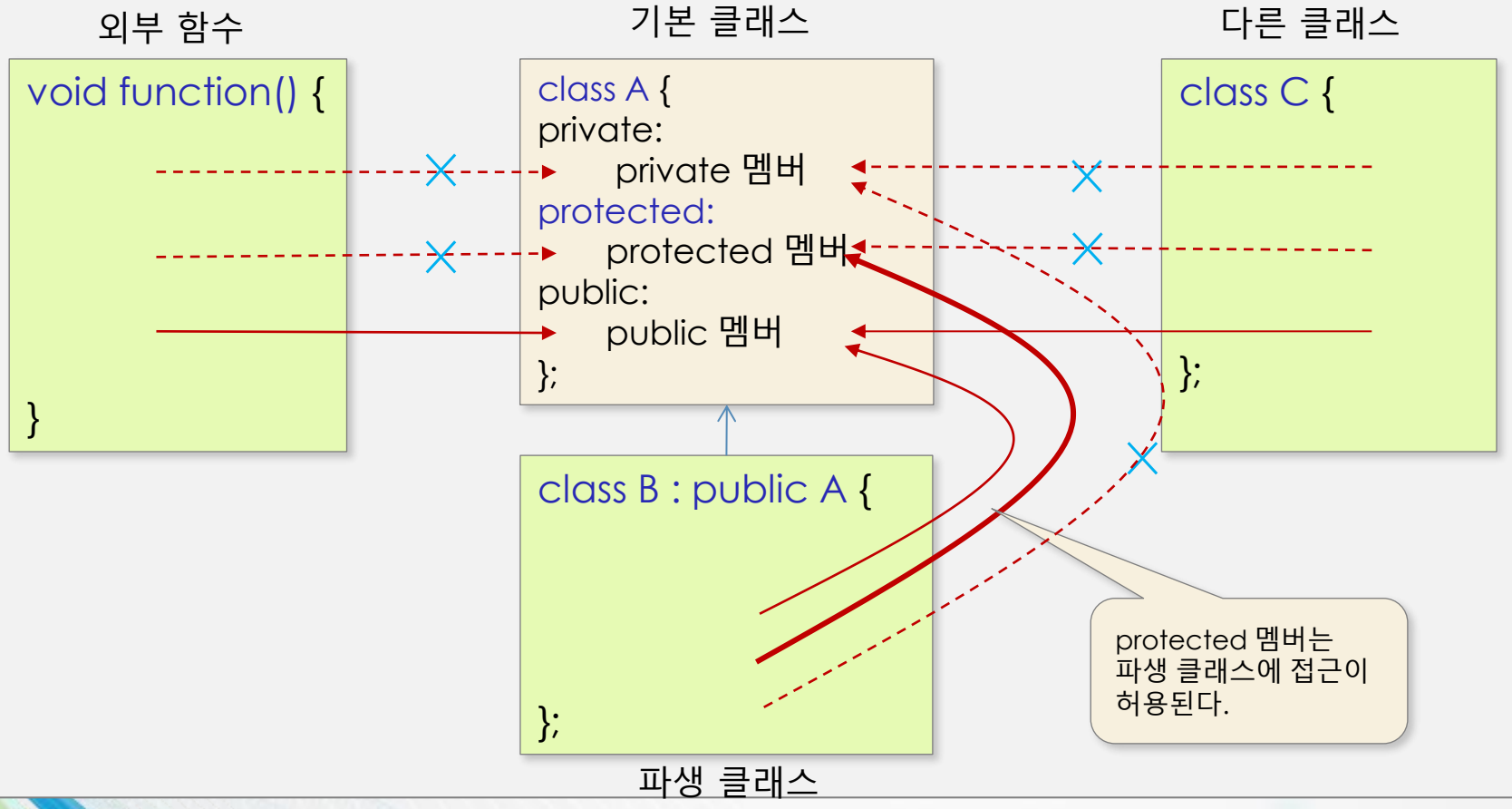
➢ 접근 지정자

- private 멤버
  - 선언된 클래스 내에서만 접근 가능
  - 파생 클래스에서도 기본 클래스의 private 멤버 직접 접근 불가
- public 멤버
  - 선언된 클래스나 외부 어떤 클래스, 모든 외부 함수에 접근 허용
  - 파생 클래스에서 기본 클래스의 public 멤버 접근 가능
- protected 멤버
  - 선언된 클래스에서 접근 가능
  - 파생 클래스에서만 접근 허용(다른 클래스나 외부 함수에서는 접근할 수 없다.)

	Public 상속	Protected 상속	Private 상속
Public 멤버	Public	Protected	private
Protected 멤버	Protected	Protected	private
Private 멤버	접근 불가	접근 불가	접근 불가

8.4 protected 접근 지정

➢ 멤버의 접근 지정에 따른 접근성



8.4 protected 접근 지정

➢ [예제 8-2] protected 멤버에 대한 접근

```
#include <iostream>
#include <string>
using namespace std;
class Point {
protected:
    int x, y; //한 점 (x,y) 좌표값
public:
    void set(int x, int y);
    void showPoint();
};
void Point::set(int x, int y) {
    this->x = x;  this->y = y;
}
void Point::showPoint() {
    cout << "(" << x << "," << y << ")" << endl;
}

class ColorPoint : public Point {
    string color;
public:
    void setColor(string color);
    void showColorPoint();
    bool equals(ColorPoint p);
};
void ColorPoint::setColor(string color) {
    this->color = color;
}
```

```
void ColorPoint::showColorPoint() {
    cout << color << ":";
    showPoint(); // Point 클래스의 showPoint() 호출
}
bool ColorPoint::equals(ColorPoint p) {
    if(x == p.x && y == p.y && color == p.color) // ①
            return true;
    else
            return false;
}
int main() {
    Point p; // 기본 클래스의 객체 생성
    p.set(2,3);             // ②
    p.x = 5;                // ③   오류
    p.y = 5;                // ④   오류
    p.showPoint();
    ColorPoint cp; // 파생 클래스의 객체 생성
    cp.x = 10;              // ⑤   오류
    cp.y = 10;              // ⑥   오류
    cp.set(3,4);  cp.setColor("Red");  cp.showColorPoint();
    ColorPoint cp2;
    cp2.set(3,4); cp2.setColor("Red");
    cout << ((cp.equals(cp2))?"true":"false");   // ⑦
}
```

8.5 상속과 생성자, 소멸자

➢ 상속 관계의 생성자와 소멸자 실행

➢ 질문 1

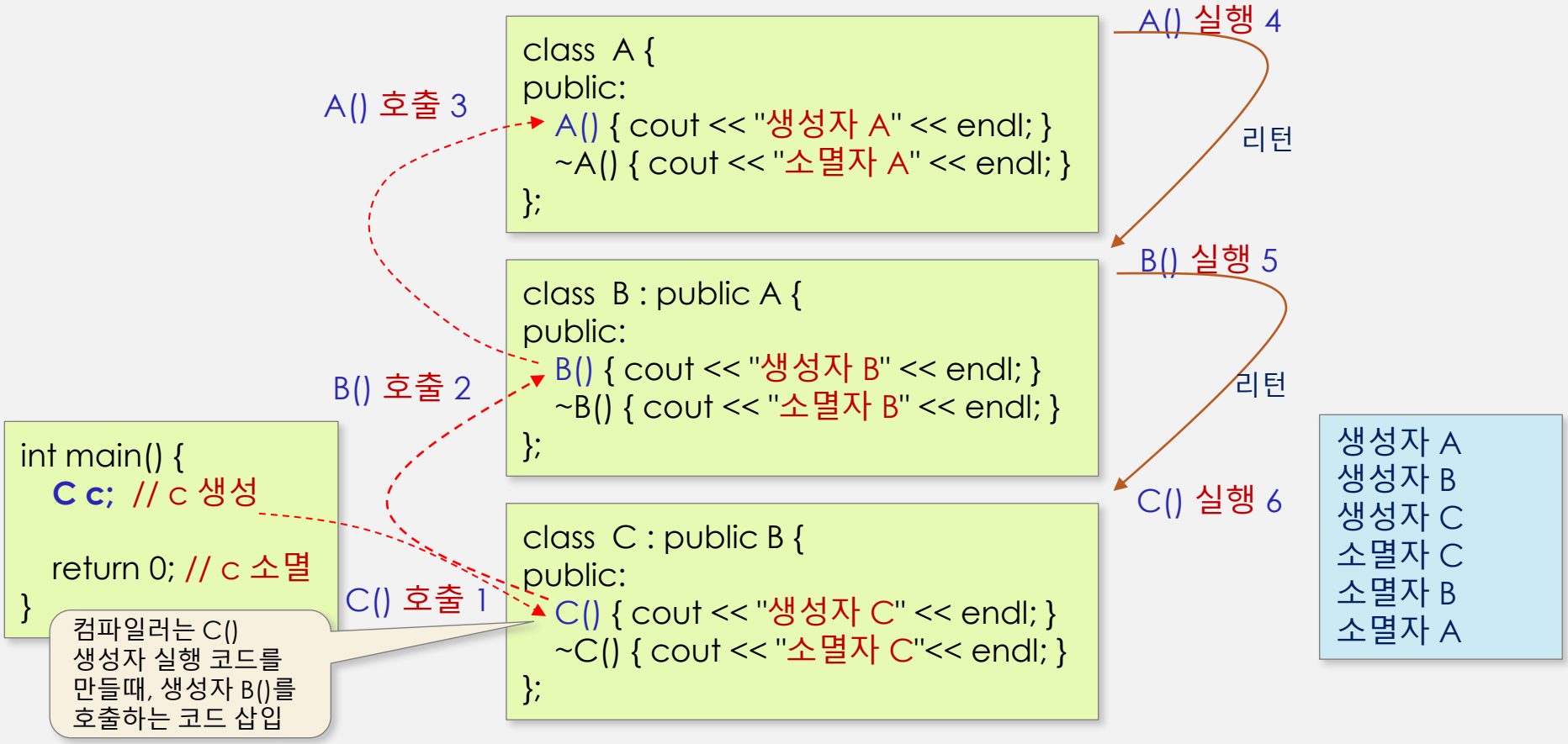
- 파생 클래스의 객체가 생성될 때 파생 클래스의 생성자와 기본 클래스의 생성자가 모두 실행되는가? 아니면 파생 클래스의 생성자만 실행되는가?
  - 답 - 둘 다 실행된다.

➢ 질문 2

- 파생 클래스의 생성자와 기본 클래스의 생성자 중에서 먼저 실행되는 생성자는?
  - 답 - 기본 클래스의 생성자가 먼저 실행된 후 파생 클래스의 생성자가 실행된다.

8.5 상속과 생성자, 소멸자

➢ 생성자 호출 관계 및 실행 순서



8.5 상속과 생성자, 소멸자

➢ 파생 클래스의 객체가 소멸될 때 소멸자의 실행 순서
- 파생 클래스의 소멸자가 먼저 실행되고
- 기본 클래스의 소멸자가 나중에 실행

8.5 상속과 생성자, 소멸자

➢ 컴파일러에 의해 묵시적으로 기본 클래스의 생성자를 선택하는 경우

- 파생 클래스를 작성한 개발자가 기본 클래스의 생성자를 선택하지 않은 경우, 컴파일러는 기본 클래스의 기본 생성자가 호출하도록 묵시적으로 선택

컴파일러는 묵시적으로 기본 클래스의 기본 생성자를 호출하도록 컴파일함

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

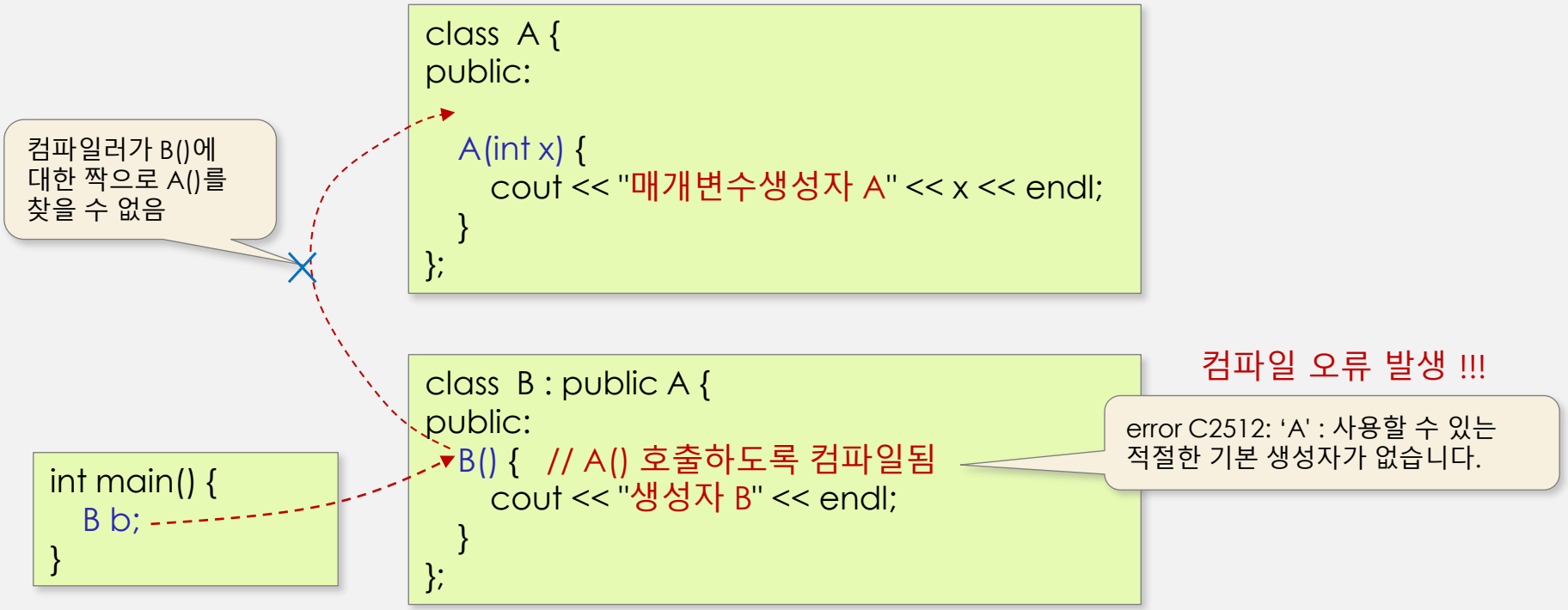
```
class  B : public A {
public:
   B() { // A() 호출하도록 컴파일됨
      cout << "생성자 B" << endl;
   }
};
```

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

```
생성자 A
생성자 B
```

8.5 상속과 생성자, 소멸자

➢ 기본 클래스에 기본 생성자가 없는 경우



8.5 상속과 생성자, 소멸자

➢ 매개 변수를 가진 파생 클래스의 생성자는 묵시적으로 기본 클래스의 기본 생성자 선택

- 파생 클래스의 매개 변수를 가진 생성자가 기본 클래스의 기본 생성자 호출

컴파일러는 묵시적으로 기본 클래스의 기본 생성자를 호출하도록 컴파일함

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

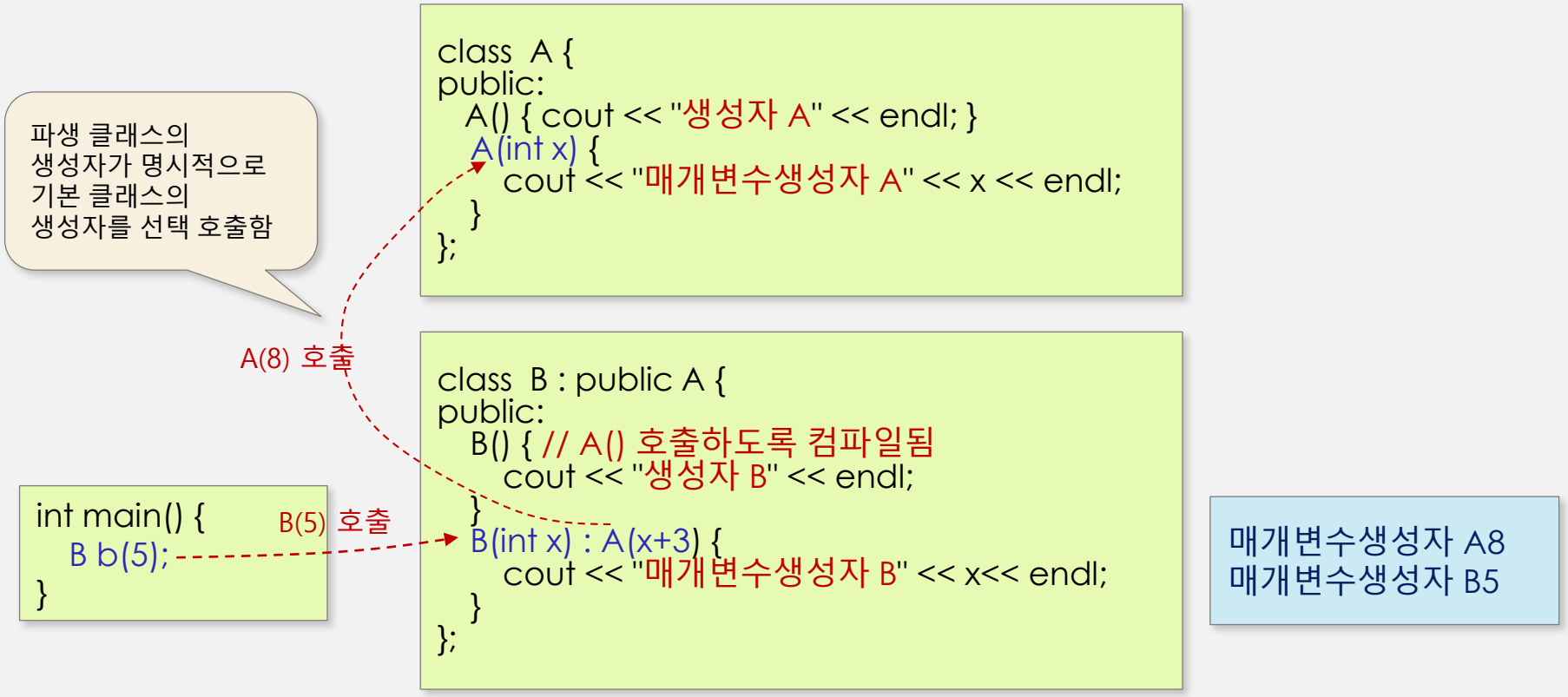
```
class  B : public A {
public:
    B() {  // A() 호출하도록 컴파일됨
        cout << "생성자 B" << endl;
    }
    B(int x) {  // A() 호출하도록 컴파일됨
        cout << "매개변수생성자 B" << x<< endl;
    }
};
```

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

```
생성자 A
매개변수생성자 B5
```

8.5 상속과 생성자, 소멸자

➢ 파생 클래스의 생성자에서 명시적으로 기본 클래스의 생성자 선택



8.5 상속과 생성자, 소멸자

➢ [예제 8-3] TV, WideTV, SmartTV 생성자 매개 변수 전달

```
#include <iostream>
#include <string>
using namespace std;
class  TV {
    int size; // 스크린 크기
public:
    TV() { size = 20; }
    TV(int size) { this->size = size; }
    int getSize() { return size; }
};
class  WideTV : public TV { // TV를 상속받는 WideTV
    bool videoIn;
public:
    WideTV(int size, bool videoIn) : TV(size) {
                this->videoIn = videoIn;
    }
    bool getVideoIn() { return videoIn; }
};
// WideTV를 상속받는 SmartTV
class  SmartTV : public WideTV {
    string ipAddr; // 인터넷 주소
public:
 SmartTV(string ipAddr, int size) : WideTV(size,true){
                this->ipAddr = ipAddr;
   }
   string getIpAddr() { return ipAddr; }
};
```

32

32

true

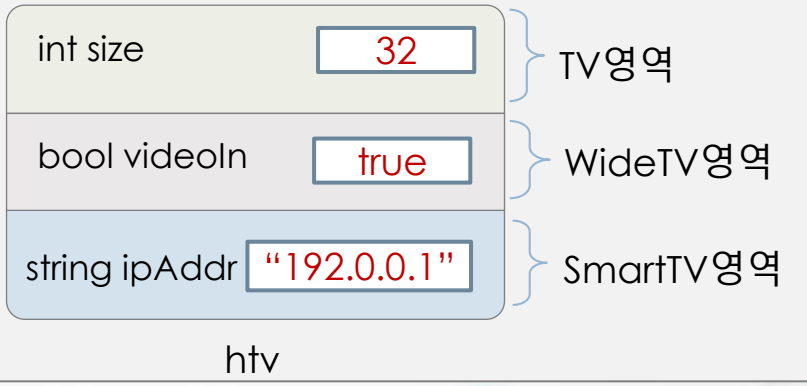
32

"192.0.0.1"

```
int main() {
// 32인치 크기에 "192.0.0.1"의 인터넷 주소를 가지는 스마트TV 객체 생성
    SmartTV  htv("192.0.0.1", 32);
    cout << "size=" << htv.getSize() << endl;
    cout << "videoIn=" << boolalpha << htv.getVideoIn() << endl;
    cout << "IP=" << htv.getIpAddr() << endl;
}
```

boolalpha는 불린 값을 true, false로 출력되게 하는 조작자

```
size=32
videoIn=true
IP=192.0.0.1
```



8.6 상속의 종류 : public, protected, private 상속

➢ 상속 지정

- 상속 선언 시 public, private, protected의 3가지 중 하나 지정

➢ 3가지 상속

- 기본 클래스의 멤버의 접근 속성을 어떻게 계승할지 지정

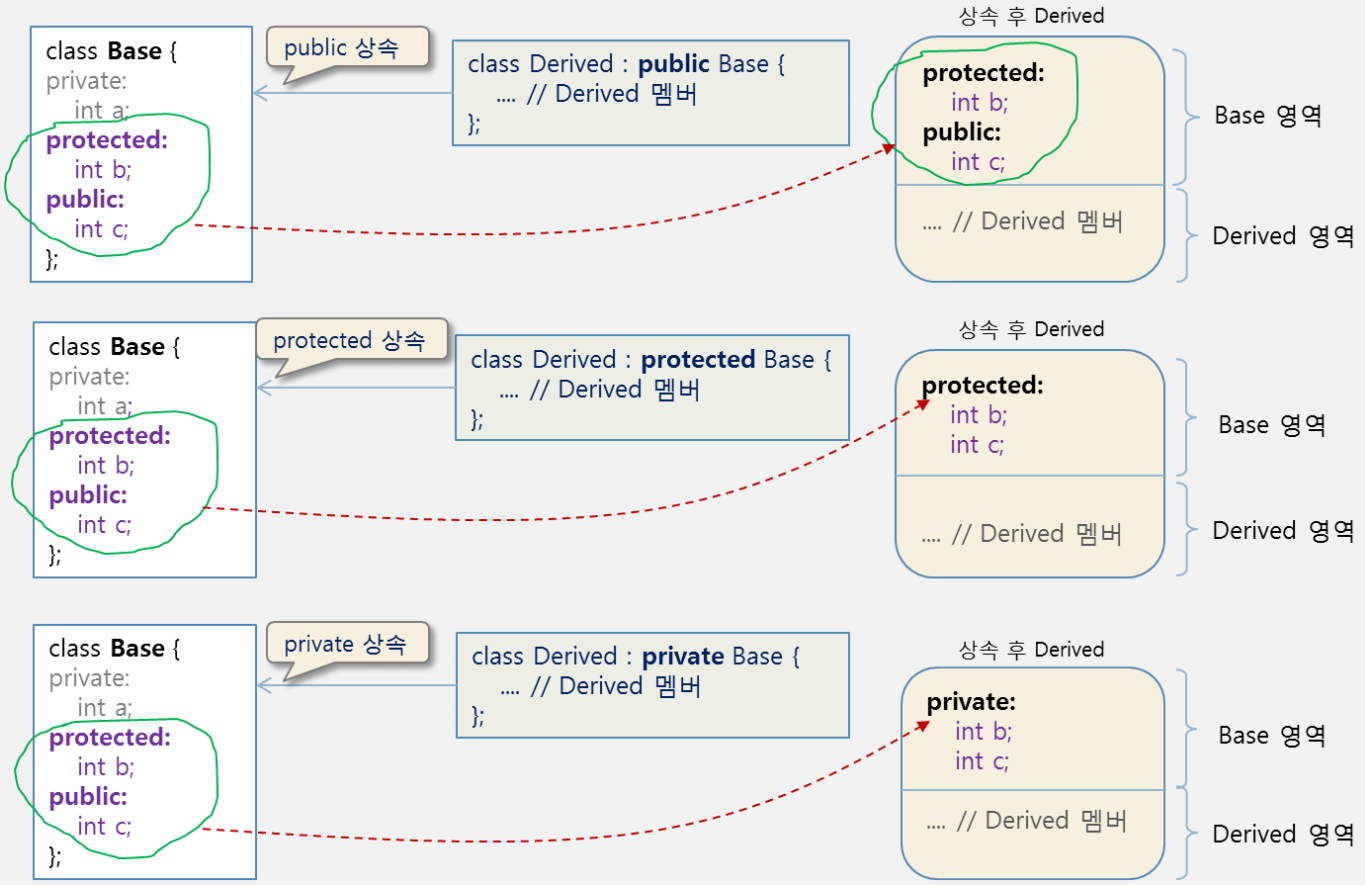
① public 상속– 기본 클래스의 protected, public 멤버 속성을 그대로 계승

② private 상속 – 기본 클래스의 protected, public 멤버를 private으로 계승

③ protected 상속 – 기본 클래스의 protected, public 멤버를 protected로 계승

8.6 상속의 종류 : public, protected, private 상속

➢ 상속 시 접근 지정에 따른 멤버의 접근 지정 속성 변화



8.6 상속의 종류 : public, protected, private 상속

➢ [예제 8-4] private 상속 사례

- 다음에서 컴파일 오류가 발생하는 부분을 찾아라.

```
#include <iostream>
using namespace std;
class Base {
   int a;
protected:
   void setA(int a) { this->a = a; }
public:
   void showA() { cout << a; }
};

class Derived : private Base {
   int b;
protected:
   void setB(int b) { this->b = b; }
public:
   void showB() { cout << b; }
};
```

```
int main() {
   Derived x;
   x.a = 5;          // ①
   x.setA(10);       // ②
   x.showA();        // ③
   x.b = 10;         // ④
   x.setB(10);       // ⑤
   x.showB();        // ⑥
}
```

컴파일 오류
①, ②, ③, ④, ⑤

8.6 상속의 종류 : public, protected, private 상속

➢ [예제 8-5] protected 상속 사례

- 다음에서 컴파일 오류가 발생하는 부분을 찾아라.

```
#include <iostream>
using namespace std;
class Base {
   int a;
protected:
   void setA(int a) { this->a = a; }
public:
   void showA() { cout << a; }
};

class Derived : protected Base {
   int b;
protected:
   void setB(int b) { this->b = b; }
public:
   void showB() { cout << b; }
};
```

```
int main() {
   Derived x;
   x.a = 5;          // ①
   x.setA(10);       // ②
   x.showA();        // ③
   x.b = 10;         // ④
   x.setB(10);       // ⑤
   x.showB();        // ⑥
}
```

컴파일 오류
①, ②, ③, ④, ⑤

8.6 상속의 종류 : public, protected, private 상속

➢ [예제 8-6] 상속이 중첩될 때 접근 지정 사례

- 다음에서 컴파일 오류가 발생하는 부분을 찾아라.

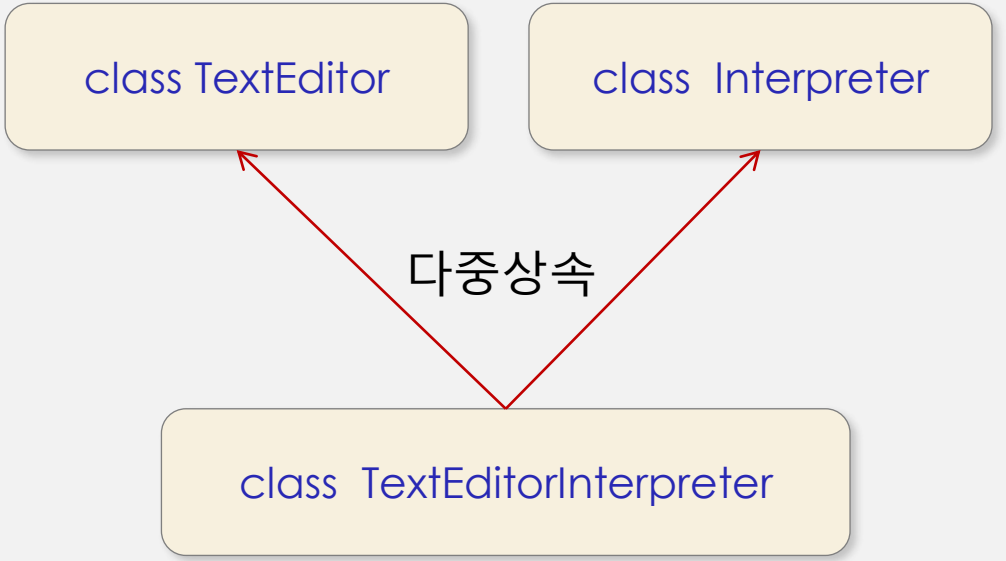
```
#include <iostream>
using namespace std;

class Base {
   int a;
protected:
   void setA(int a) { this->a = a; }
public:
   void showA() { cout << a; }
};

class Derived : private Base {
   int b;
protected:
   void setB(int b) { this->b = b; }
public:
   void showB() {
      setA(5);          // ①
      showA();          // ②
      cout << b;
   }
};
```

```
class GrandDerived : private Derived {
   int c;
protected:
   void setAB(int x) {
      setA(x);          // ③
      showA();          // ④
      setB(x);          // ⑤
   }
};
```

컴파일 오류
③, ④

8.7 다중 상속

➢ 기기의 컨버전스와 C++의 다중 상속



8.7 다중 상속

➢ 다중 상속 선언 및 멤버 호출

다중 상속 선언

```
class MP3 {
public:
        void play();
        void stop();
};
class MobilePhone {
public:
        bool sendCall();
        bool receiveCall();
        bool sendSMS();
        bool receiveSMS();
};
// 다중 상속 선언
class MusicPhone : public MP3,
                   public MobilePhone {
public:
        void dial();
};
```

다중 상속 활용

```
void MusicPhone::dial() {
      play();              // mp3 음악을 연주시키고
      sendCall();          // 전화를 건다.
}
```

다중 상속 활용

```
int main() {
      MusicPhone hanPhone;
      hanPhone.play();  // MP3의 멤버 play() 호출

      // MobilePhone의 멤버 sendSMS() 호출
      hanPhone.sendSMS();
}
```

8.7 다중 상속

➢ [예제 8-7] Adder와 Subtractor를 다중 상속 받는 Calculator 클래스 작성

- Adder와 Subtractor를 다중 상속받는 Calculator를 작성하라.

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

class Adder {
protected:
        int add(int a, int b) { return a+b; }
};
class Subtractor {
protected:
        int minus(int a, int b) { return a-b; }
};
```

```
class Calculator : public Adder, public Subtractor {
public:
        int calc(char op, int a, int b);
};
int Calculator::calc(char op, int a, int b) {
        int res=0;
        switch(op) {
                case '+' : res = add(a, b); break;
                case '-' : res = minus(a, b); break;
        }
        return res;
}
```

```
2 + 4 = 6
100 – 8 = 92
```

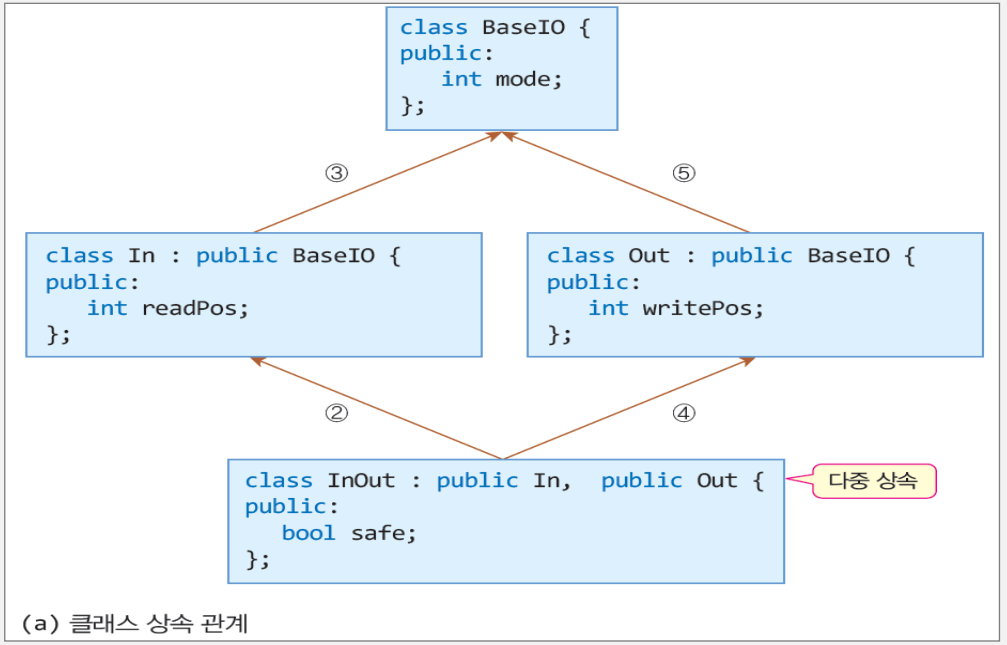
```
int main() {
        Calculator handCalculator;
        cout << "2 + 4 = " << handCalculator.calc('+', 2, 4) << endl;
        cout << "100 - 8 = " << handCalculator.calc('-', 100, 8) << endl;
}
```

8.7 다중 상속

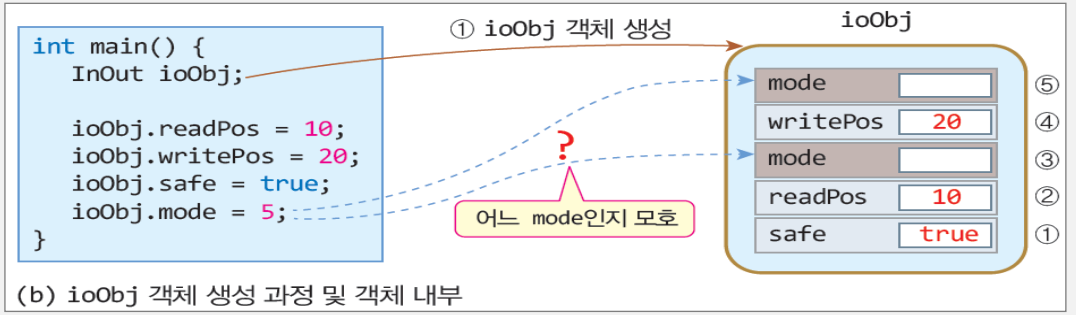
➢ 다중 상속의 문제점
- 기본 클래스 멤버의 중복 상속

- Base의 멤버가 이중으로 객체에 삽입되는 문제점

- 동일한 x를 접근하는 프로그램이 서로 다른 x에 접근하는 결과를 인해 잘못된 실행 오류가 발생된다.



(a) 클래스 상속 관계



(b) ioObj 객체 생성 과정 및 객체 내부

8.8 가상 상속

➢ 다중 상속으로 인한 기본 클래스 멤버의 중복 상속 해결

➢ 가상 상속
- 파생 클래스의 선언문에서 기본 클래스 앞에 virtual로 선언
- 파생 클래스의 객체가 생성될 때 기본 클래스의 멤버는 오직 한 번만 생성
  - 기본 클래스의 멤버가 중복하여 생성되는 것을 방지

```
class In : virtual public BaseIO {  // In 클래스는 BaseIO 클래스를 가상 상속함
...
};
class Out : virtual public BaseIO {  // Out 클래스는 BaseIO 클래스를 가상 상속함
...
};
```

8.8 가상 상속

➢ 가상 상속으로
다중 상속의 모호성 해결