객체지향프로그래밍

Lecture 12 : 상속

충북대 소프트웨어학부

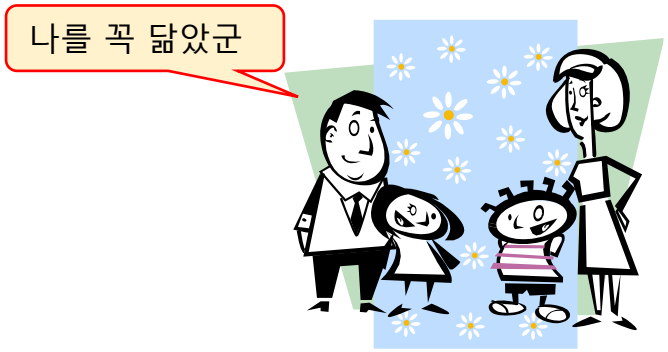
이 태 겸 (showm321@gmail.com)

목차

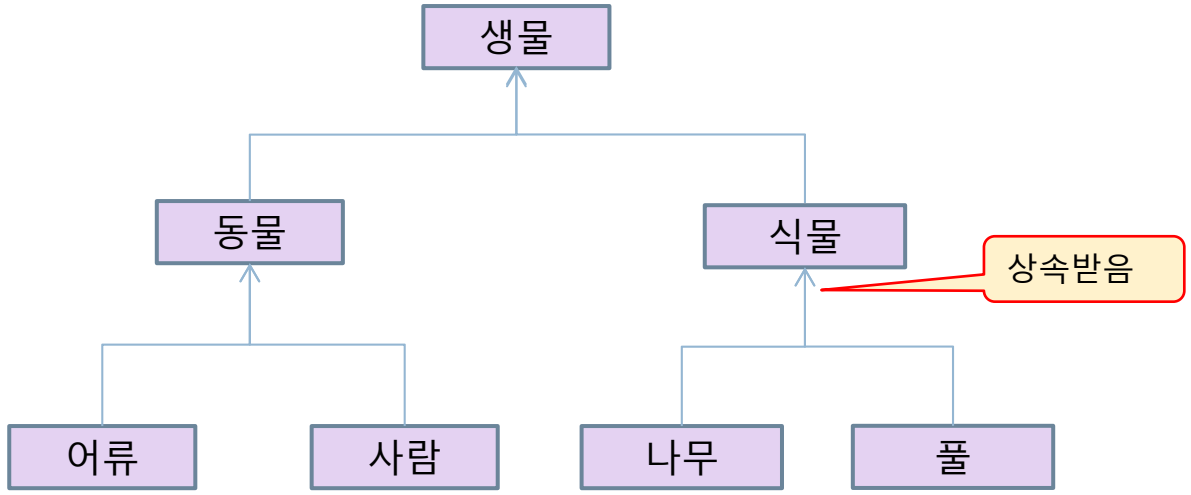
유전적 상속과 객체지향 상속

❖ 유전적 상속

- 어류와 사람은 동물(움직임, 식이를 통해 에너지획득)과 생물(살아있음)의 **속성**을 모두 가짐
- 나무와 풀은 식물(움직이지 못함, 광합성을 통해 에너지 획득)과 생물(살아있음)의 **속성**을 모두 가짐
- 단, 사람은 광합성을 하지 못하고 나무는 식이를 하지 못함
- 부모와 자식은 **생김새(특징)가** 닮음



유전적 상속 == 객체지향 상속



유전적 상속과 관계된 생물 분류

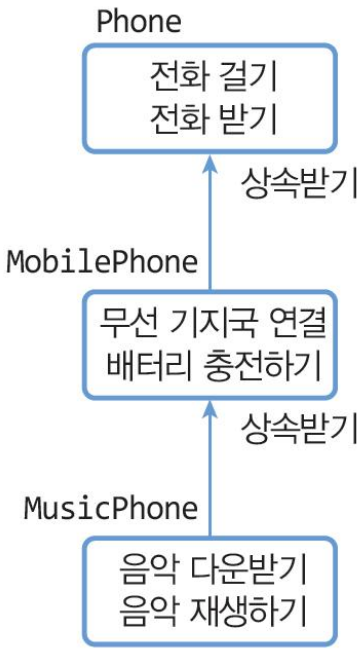
C++에서의 상속(Inheritance)

❖C++에서의 상속이란?

- **클래스 사이에서 상속 관계 정의**
  - 객체 사이에는 상속 관계 없음
- **기존 클래스의 기능을 물려 받아 새로운 클래스를 정의하는 것**
  - 기본 클래스 : 상속을 해주는 클래스, 부모가 되는 클래스, 공통적이고 일반적인 특징을 제공
  - 파생 클래스 : 상속을 받는 클래스, 자식이 되는 클래스, 일반적인 특징+자식만의 새롭고 구체적인 특징
    - 기본 클래스의 속성과 기능을 물려받고 자신 만의 속성과 기능을 추가하여 작성
- 기본 클래스에서 **파생 클래스로 갈수록 클래스의 개념이 구체화 (파생 클래스만의 구체적인 특징)**
- 다중 상속을 통한 클래스의 재활용성 높임

상속을 통해 파생 클래스를 정의할 때,
기본 클래스에 구현되어 있는 기능은 파생 클래스에 다시 구현할 필요가 없다.

상속의 표현



상속 관계 표현

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

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

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

Phone을 상속받는다.

MobilePhone을 상속받는다.

C++로 상속 선언

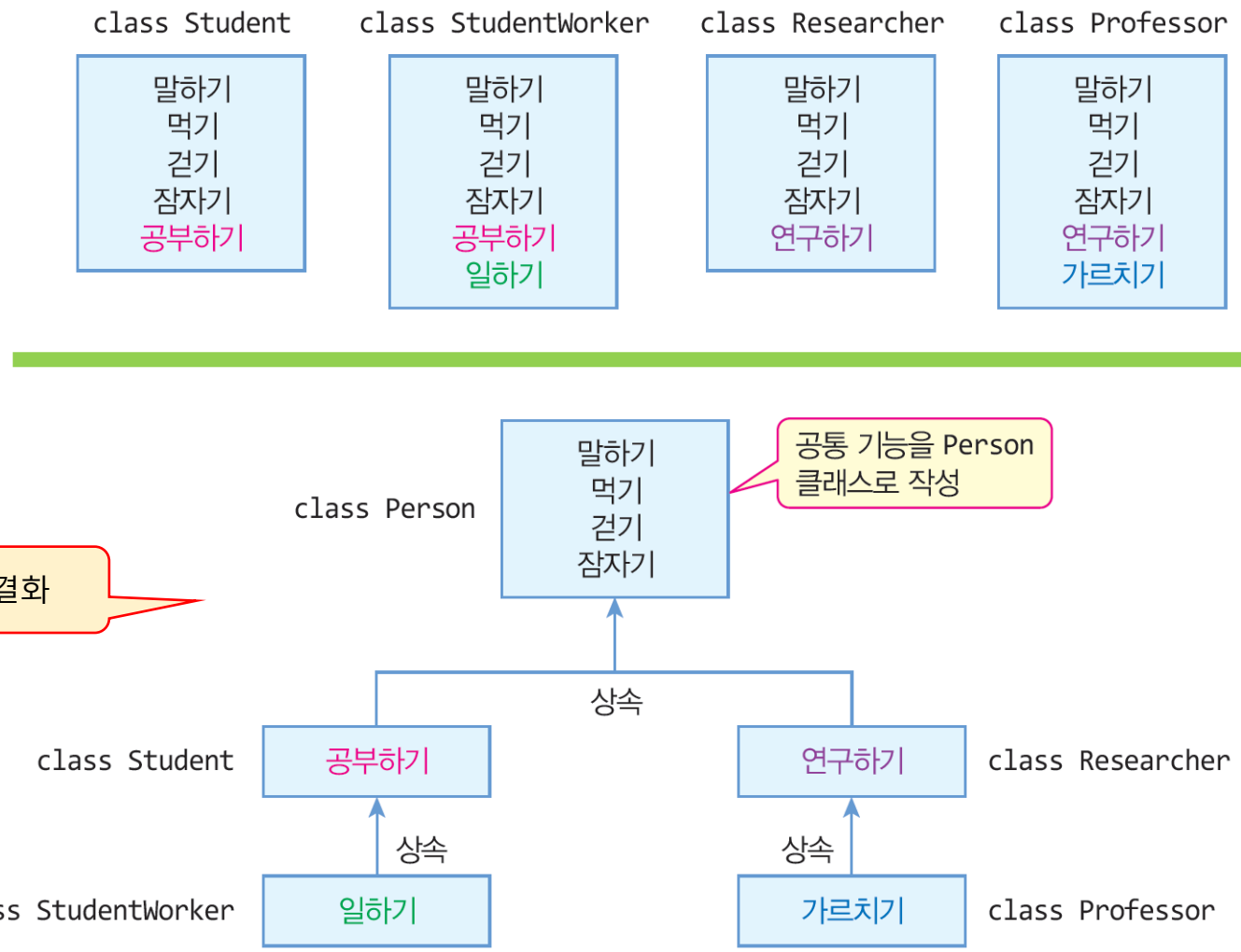
전화기

휴대 전화기

음악 기능 전화기

상속의 목적 및 장점

- 간결한 클래스 작성
- 클래스 간의 계층적 분류 및 관리의 용이함
- 클래스 재사용과 확장을 통한 소프트웨어 생산성 향상



목차

상속 선언

❖ 파생 클래스의 정의

- 클래스 이름 다음에 콜론(:)을 쓰고 public 키워드와 함께 기본 클래스 이름을 적어줌

```
class 파생클래스이름 : public 기본클래스이름
{
};
```

❖ 파생 클래스의 멤버

- **상속받는 멤버 변수는 다시 정의하지 않음**
- 새로 추가된 멤버 변수나 멤버 함수 정의
- **상속 받은 멤버 함수 중에서 기본 클래스와 다르게 처리할 멤버 함수를 재정의(overriding)**

Point 클래스를 상속받는 ColorPoint 클래스 만들기

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

// 2차원 평면에서 한 점을 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;
   }
};
```

```
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
}
```

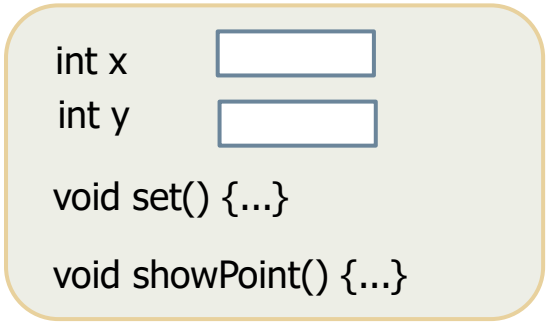
```
Red:(3,4)
```

파생 클래스의 객체 구성

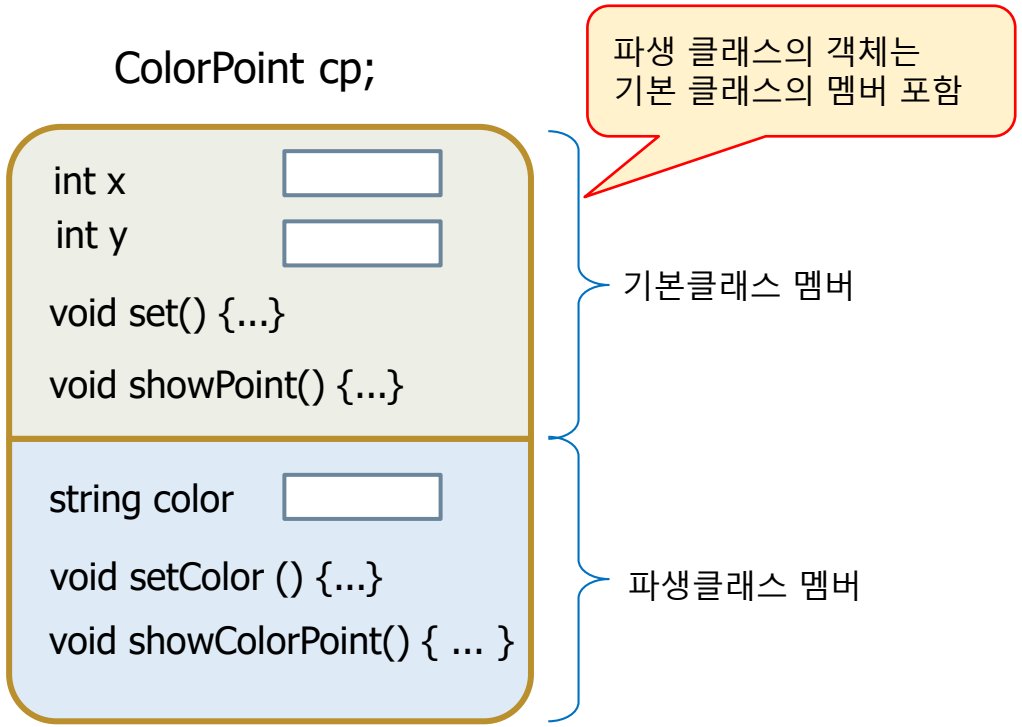
```
class Point {
   int x, y; // 한 점 (x,y) 좌표 값
public:
   void set(int x, int y);
   void showPoint();
};
```

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

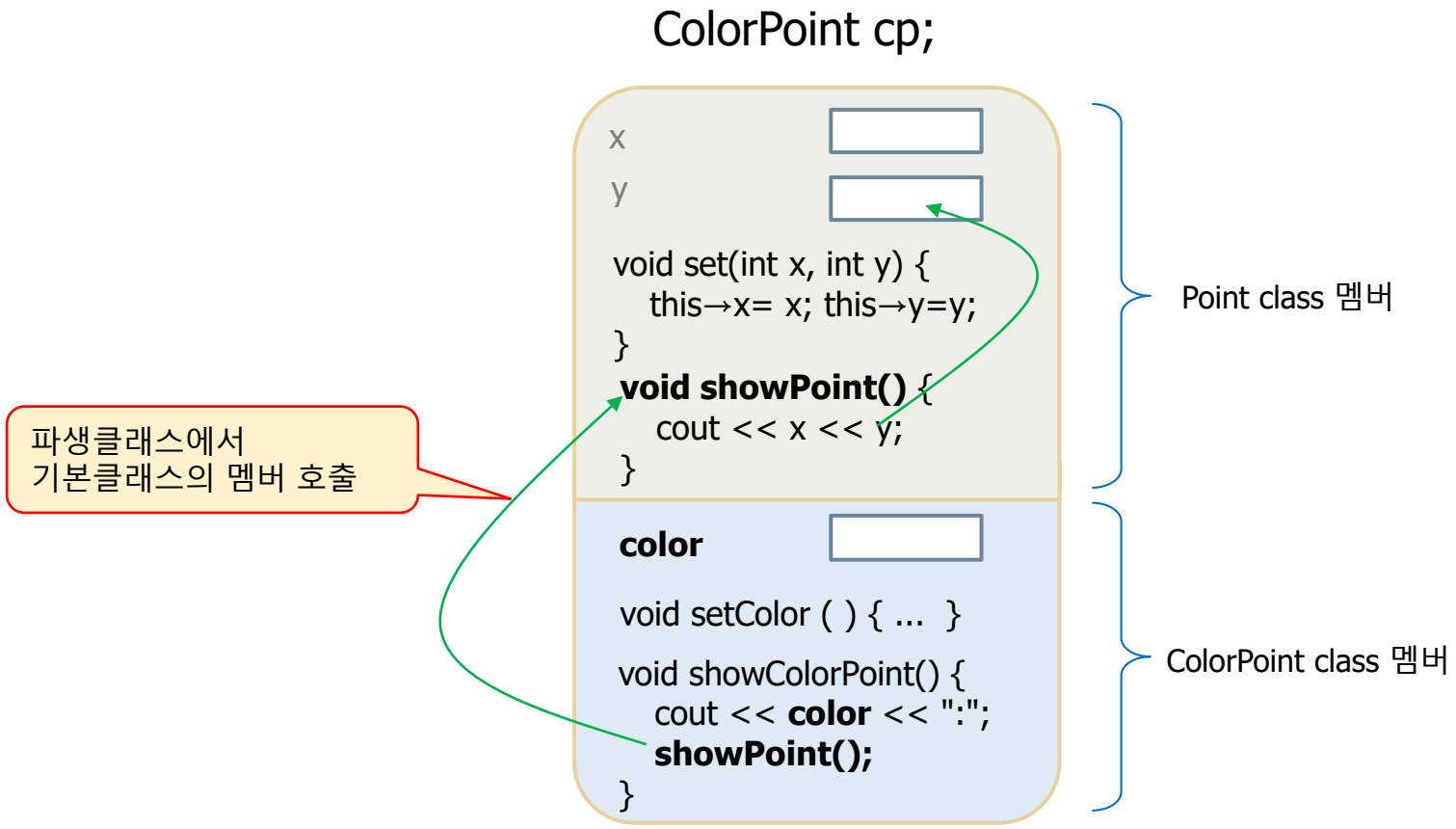
Point p;



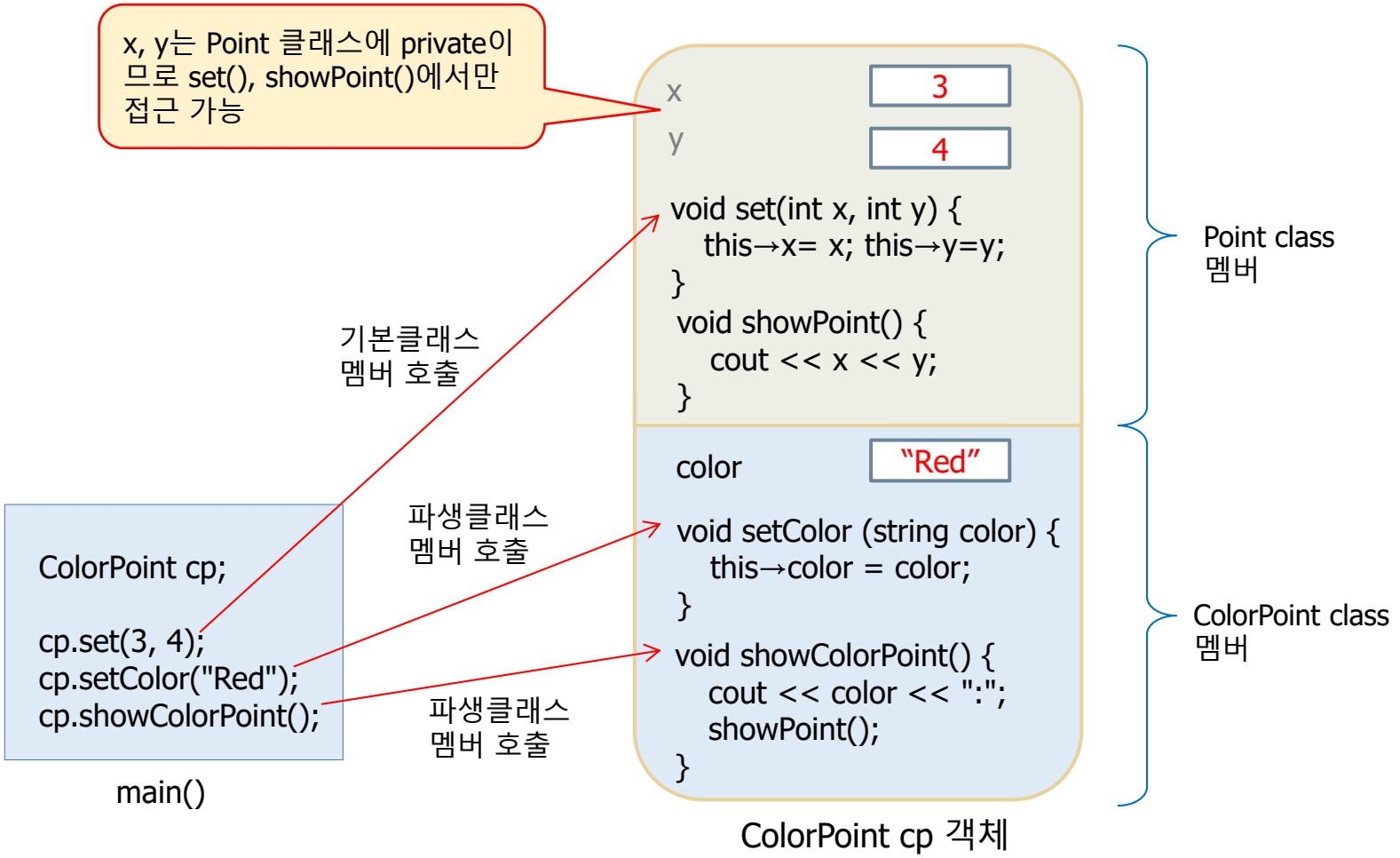
ColorPoint cp;



파생 클래스에서 기본 클래스 멤버 접근



외부에서 파생 클래스 객체에 대한 접근

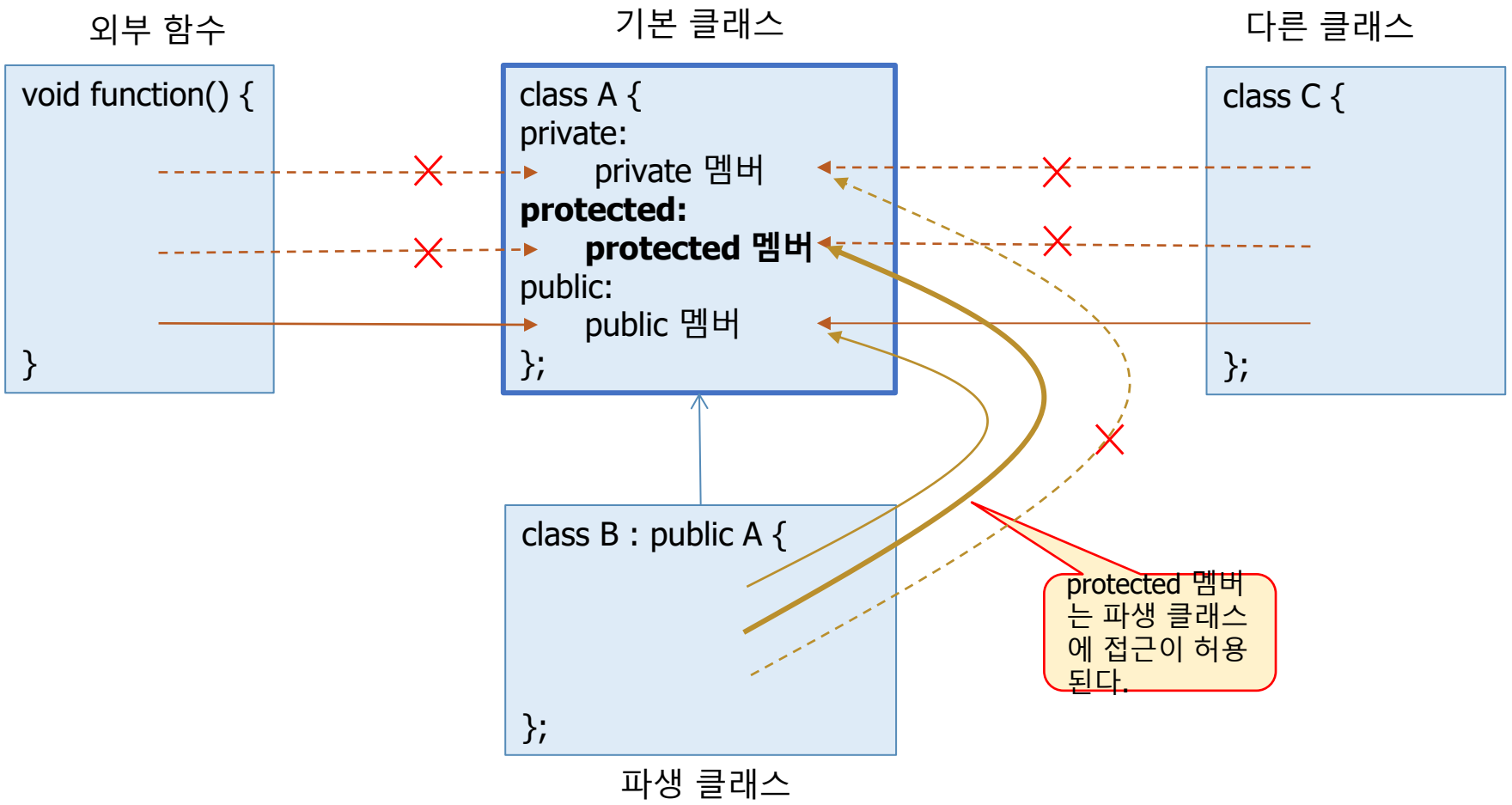


protected 접근 지정

❖ 접근 지정자

- private 멤버
  - 선언된 클래스 **내에서만 접근 허용 (외부접근 불가)**
  - 파생 클래스에서 기본 클래스의 private 멤버 직접 접근 불가
- public 멤버
  - 선언된 클래스 **내,외부 클래스 접근 허용**
  - 파생 클래스에서 기본 클래스의 public 멤버 접근 가능
- protected 멤버
  - 선언된 클래스 **내부** 그리고 **파생 클래스에서 접근 허용**
  - 파생 클래스가 아닌 다른 클래스나 외부 함수에서는 protected 멤버를 접근 불가

멤버의 접근 지정에 따른 접근성



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
    p.y = 5;                       // ④
    p.showPoint();

    ColorPoint cp; // 파생 클래스의 객체 생성
    cp.x = 10;                     // ⑤
    cp.y = 10;                     // ⑥
    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
}
```

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
    p.y = 5;                       // ④
    p.showPoint();

    ColorPoint cp; // 파생 클래스의 객체 생성
    cp.x = 10;                     // ⑤
    cp.y = 10;                     // ⑥
    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
}
```

오류

오류

오류

오류

목차

상속 관계의 생성자와 소멸자 실행

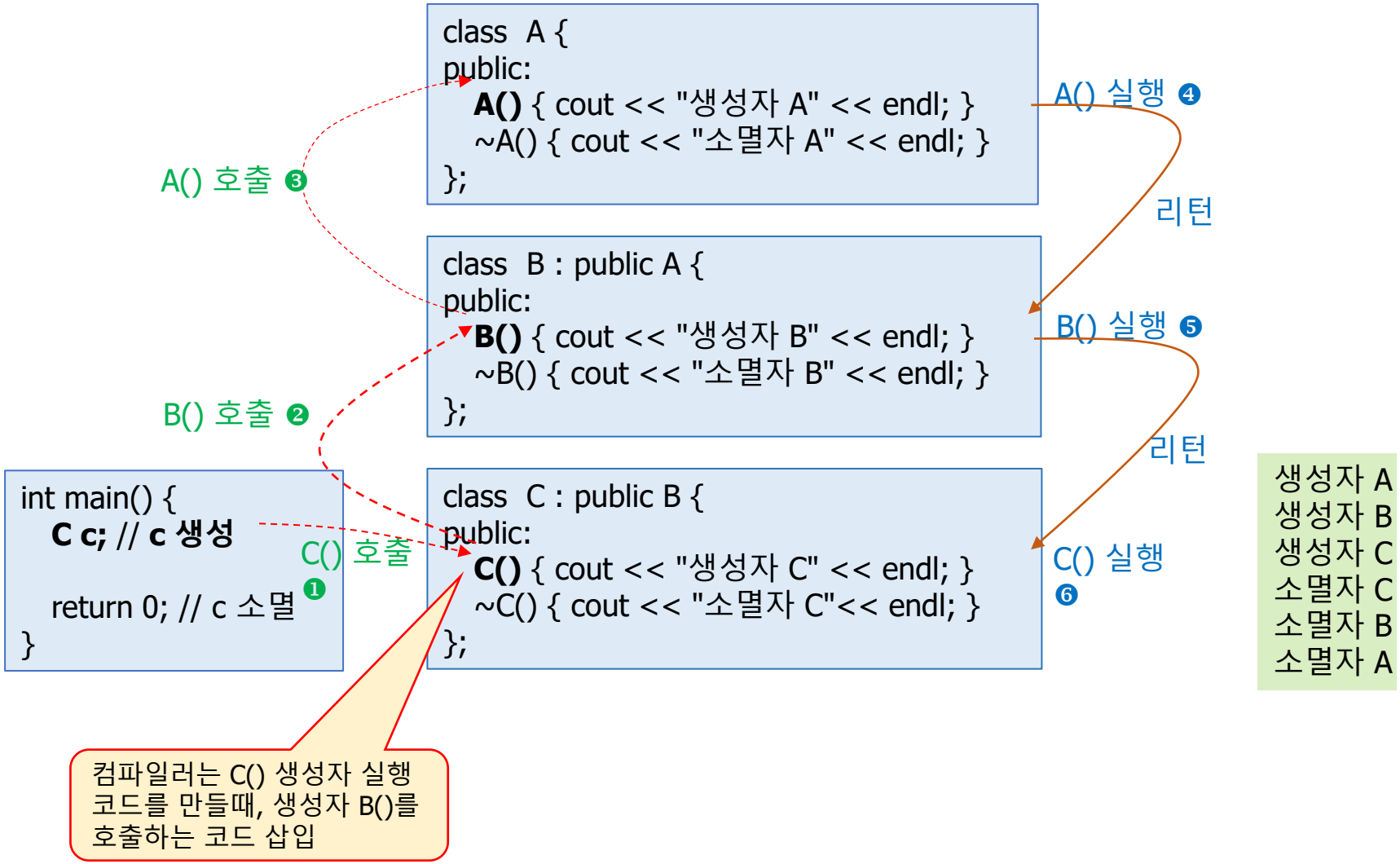
❖ 질문 1

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

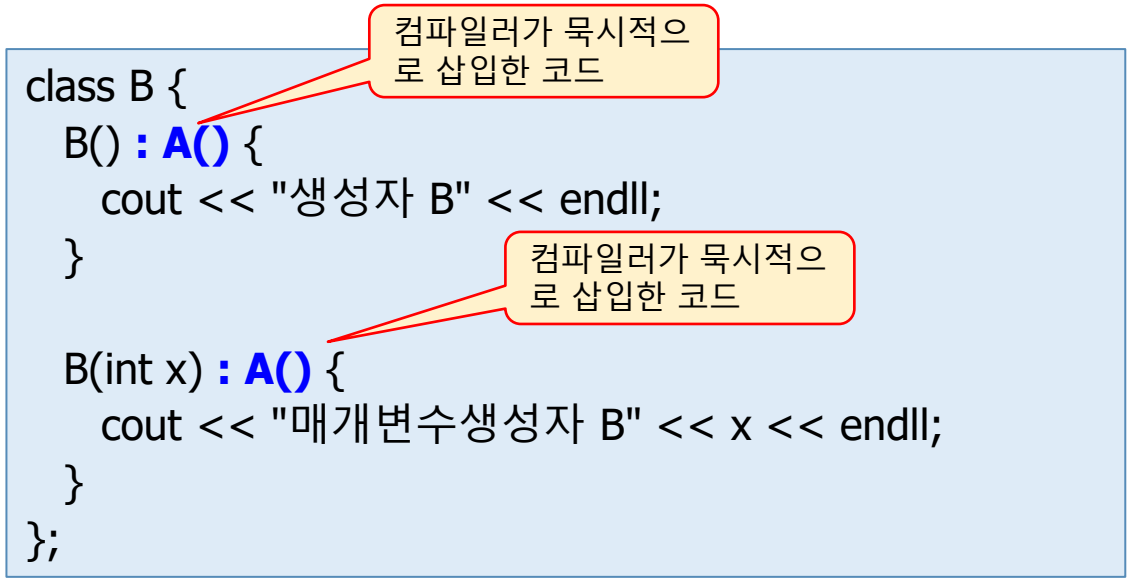
❖ 질문 2

- 파생 클래스의 생성자와 기본 클래스의 생성자 중에서 어떤 생성자가 먼저 실행되는가?
- 답 - 기본 클래스의 생성자가 먼저 실행된 후 파생 클래스의 생성자가 실행된다.

생성자 호출 관계 및 실행 순서



컴파일러의 디폴트 생성자 호출 코드 삽입

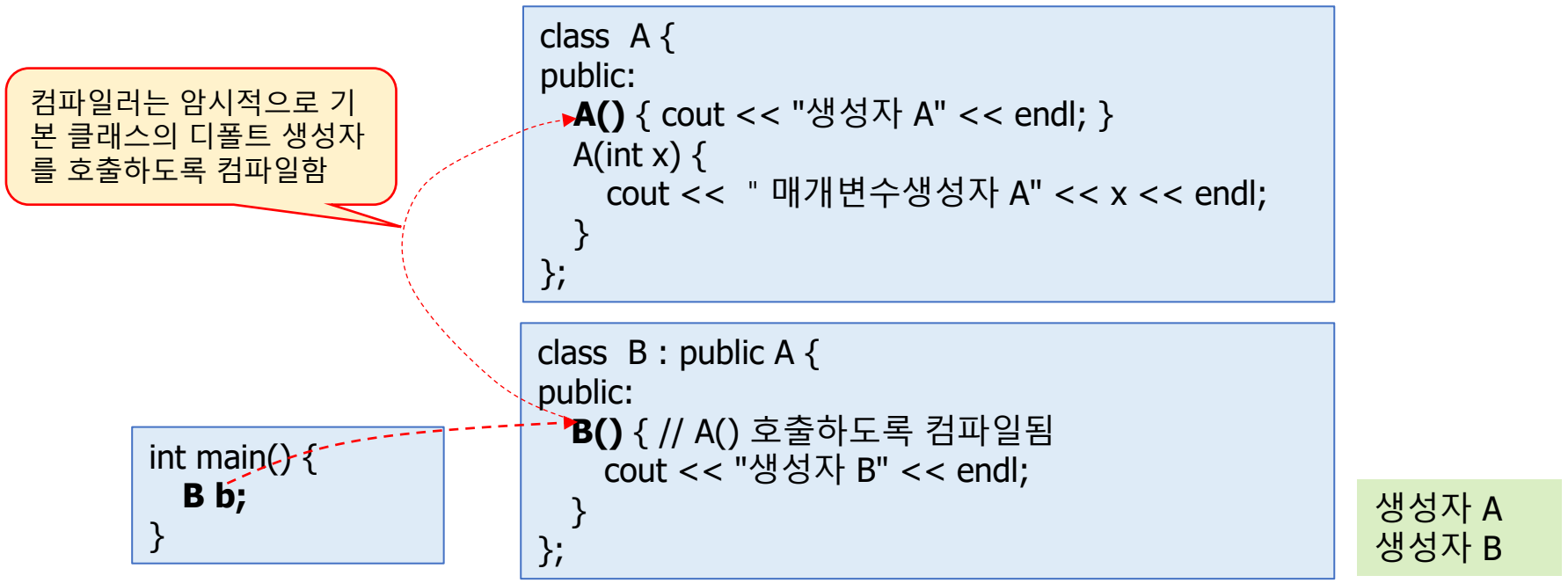


소멸자의 실행 순서

❖ 파생 클래스의 객체가 소멸될 때
- 파생 클래스의 소멸자가 먼저 실행되고
- 기본 클래스의 소멸자가 나중에 실행

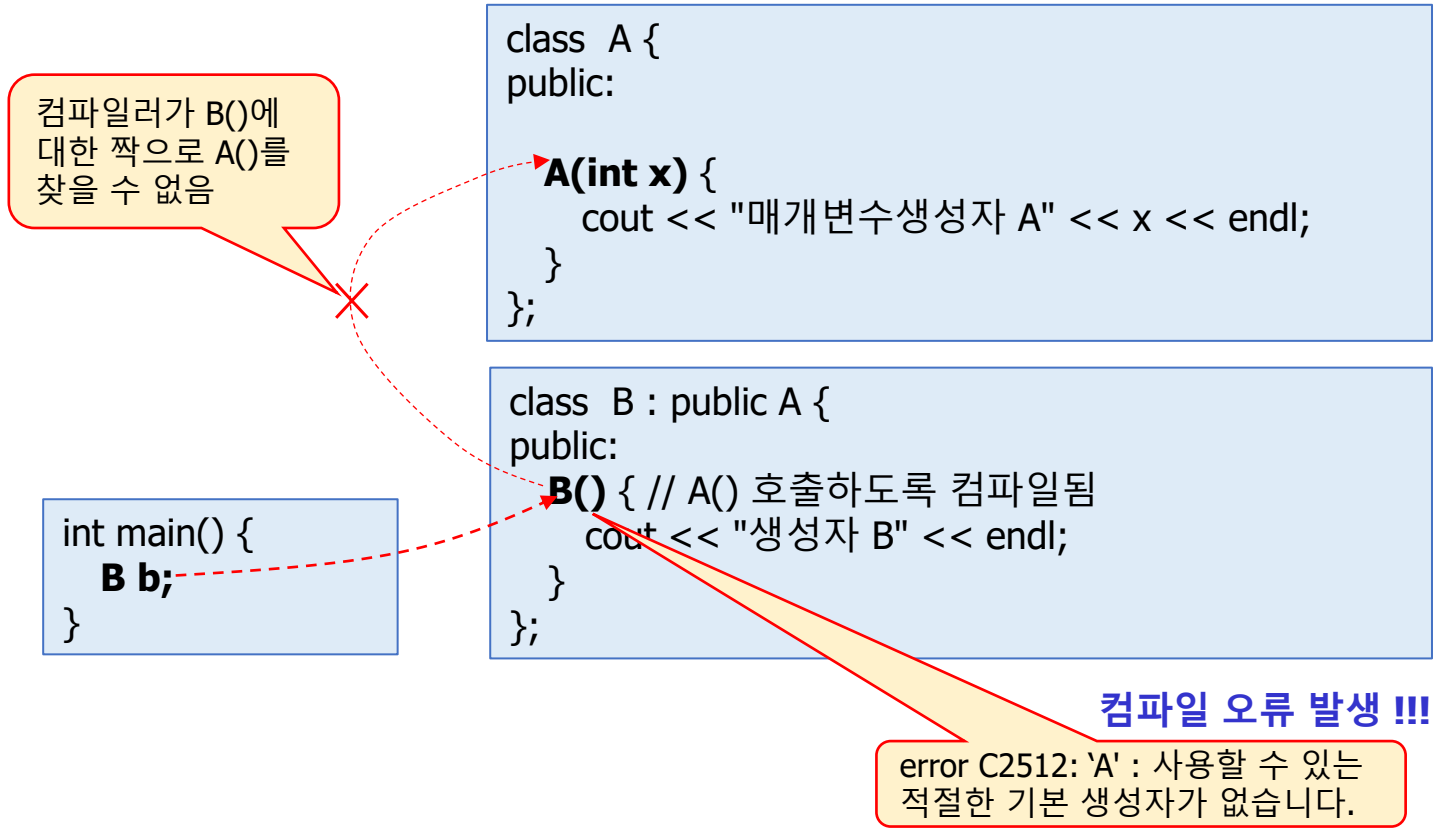
파생 클래스의 생성자 호출 사례1

❖ 기본 클래스의 디폴트 생성자의 암시적 호출



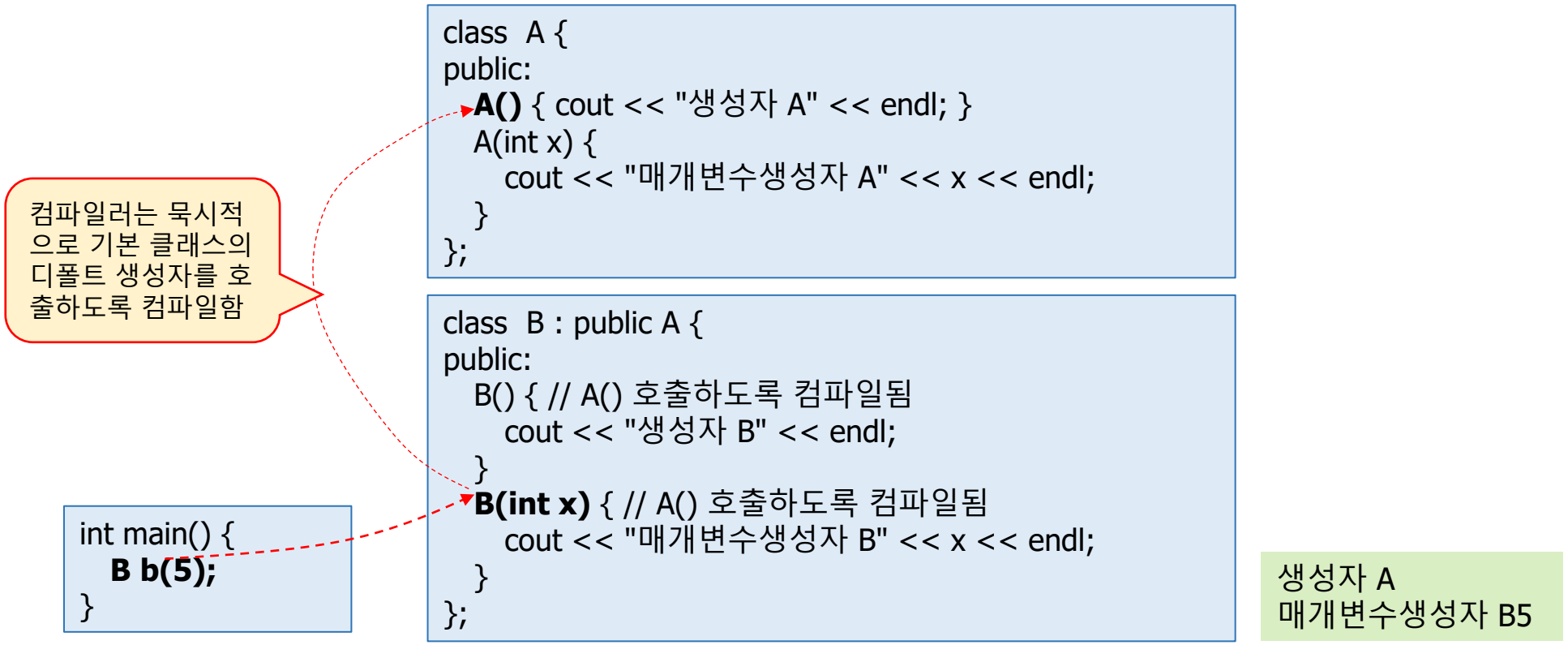
파생 클래스의 생성자 호출 사례2

❖기본 클래스에 디폴트 생성자가 없는 경우



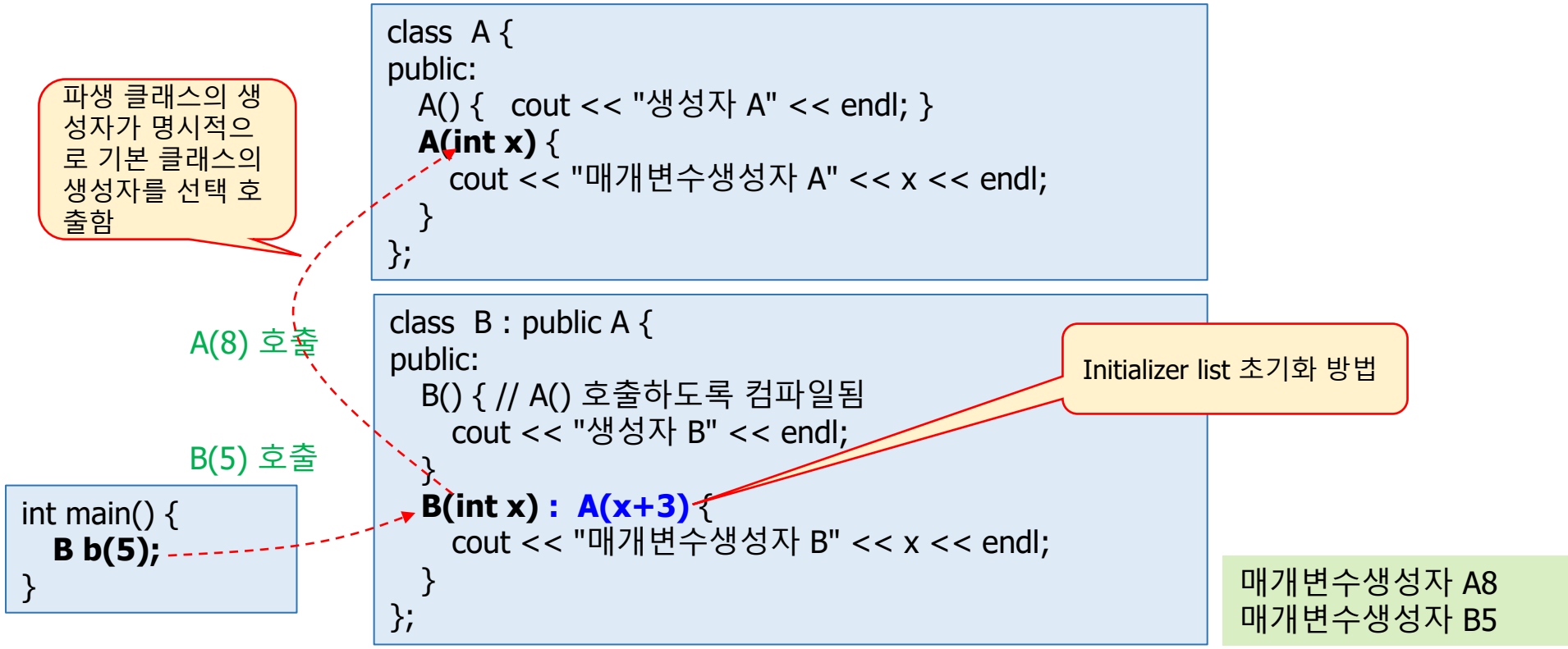
파생 클래스의 생성자 호출 사례3

❖파생 클래스의 인자 있는 생성자가 기본 클래스의 디폴트 생성자 호출



파생 클래스의 생성자 호출 사례4

❖파생 클래스의 생성자에서 명시적으로 기본 클래스의 특정한 생성자의 명시적 호출



파생 클래스의 소멸자

❖ 항상 내부적으로 기본 클래스의 소멸자 호출

- 파생 클래스의 객체가 소멸될 때 파생 클래스 소멸자는 내부적으로 기본 클래스의 소멸자를 호출함.
- 소멸자는 오버로딩이 되지 않으므로 인자 없는 소멸자 사용

목차

접근변경자

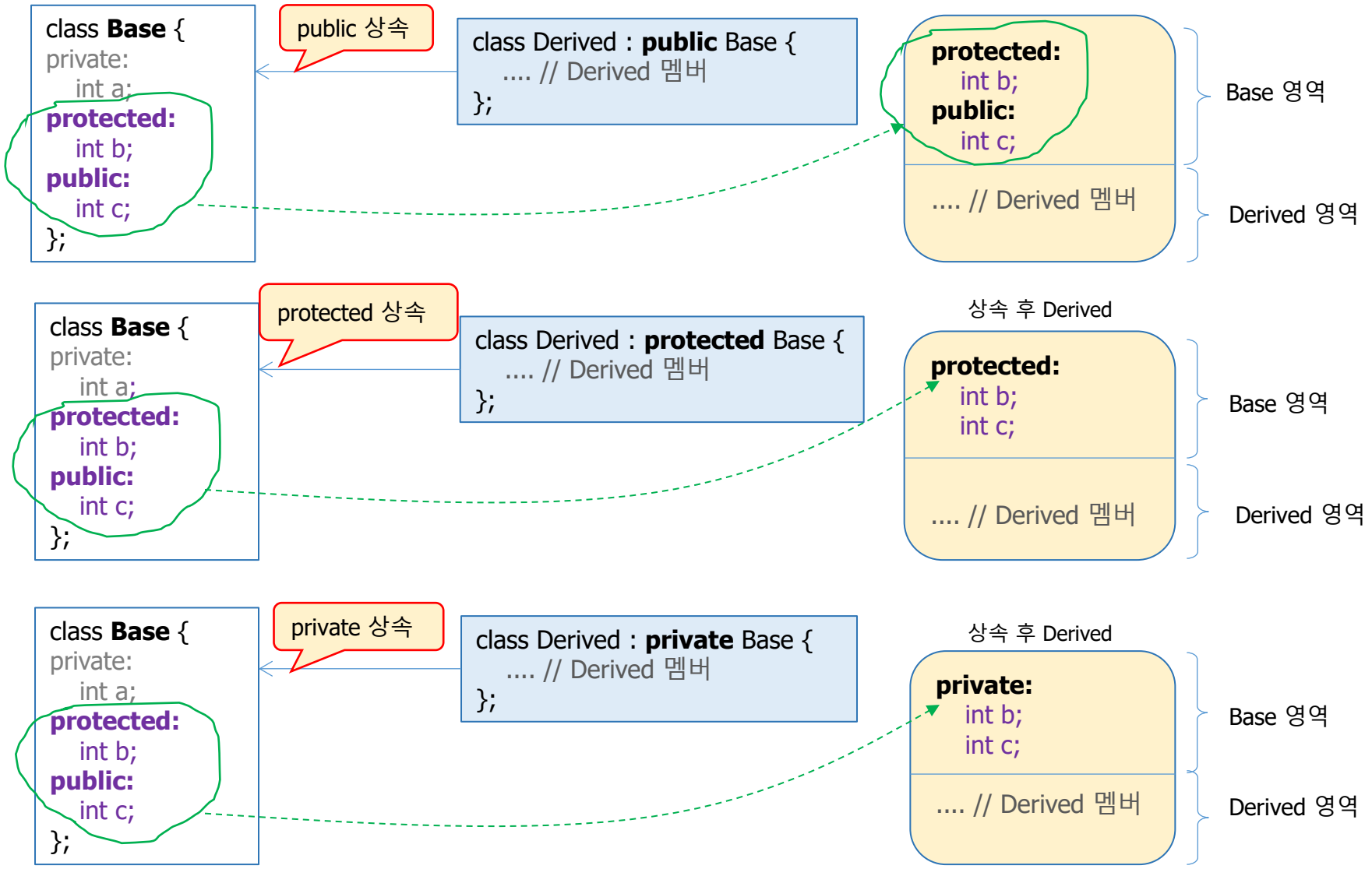
❖ 접근변경자

- 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

접근변경자	기본 클래스의 private 멤버	기본 클래스의 protected 멤버	기본 클래스의 public 멤버
private 상속	파생클래스에서 접근 불가	파생클래스의 private 멤버로 간주	파생클래스의 private 멤버로 간주
protected 상속	파생클래스에서 접근 불가	파생클래스의 protected 멤버로 간주	파생클래스의 protected 멤버로 간주
public 상속	파생클래스에서 접근 불가	파생클래스의 protected 멤버로 간주	파생클래스의 public 멤버로 간주

- 주로 public 상속 사용
- 접근 변경자도 접근 지정자처럼 디폴트로 private을 사용하므로 반드시 public을 명시적으로 지정해야 함.

상속 시 접근 지정에 따른 멤버의 접근 지정 속성 변화



private 상속 사례

```
#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; }
   void showC() { cout << a; }
};
```

```
int main() {
   Derived x;
   x.a = 5;          // ① private 멤버접근
   x.setA(10);       // ② private 멤버접근
   x.showA();        // ③ private 멤버접근
   x.b = 10;         // ④ private 멤버접근
   x.setB(10);       // ⑤ protected 멤버접근
   x.showB();        // ⑥ 정상실행
   x.showC();        // ⑦ 자식클래스에서 부모클래스의 private 멤버접근 불가
}
```

protected 상속 사례

```
#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
}
```

상속이 중첩될 때 접근 지정 사례

```
#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;
   }
};
```

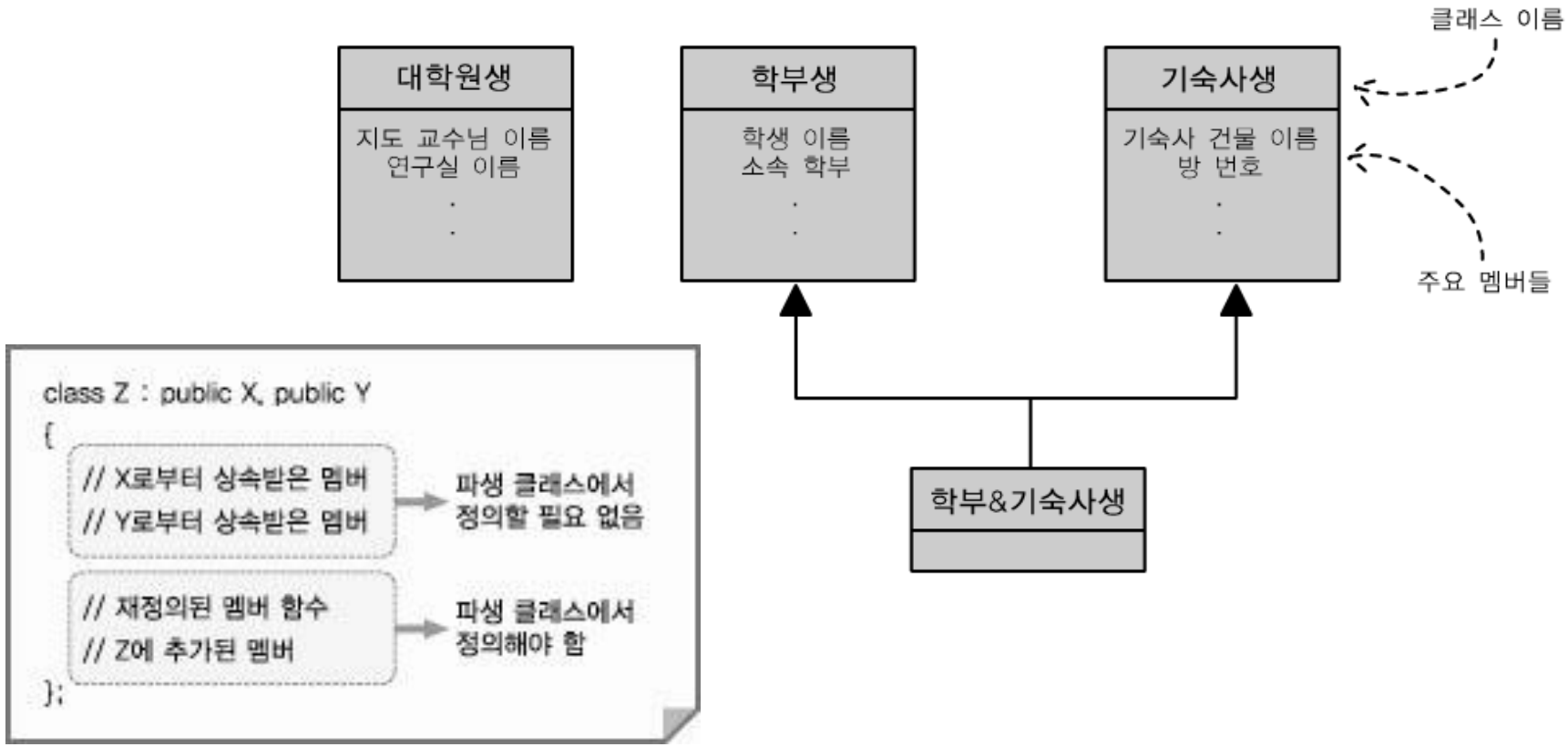
```
class GrandDerived : private Derived {
   int c;
protected:
   void setAB(int x) {
      setA(x);              // ③
      showA();              // ④
      setB(x);              // ⑤
   }
};
```

목차

다중 상속의 필요성

❖2개의 기본 클래스를 상속 받을 필요가 있는 경우

- 모든 기본 클래스의 멤버를 상속 받음



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();
};

class MusicPhone : public MP3, public MobilePhone { // 다중 상속 선언
public:
    void dial();
};
```

상속받고자 하는 기본 클래스를 나열한다.

다중 상속 활용

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

MP3::play() 호출

MobilePhone::sendCall() 호출

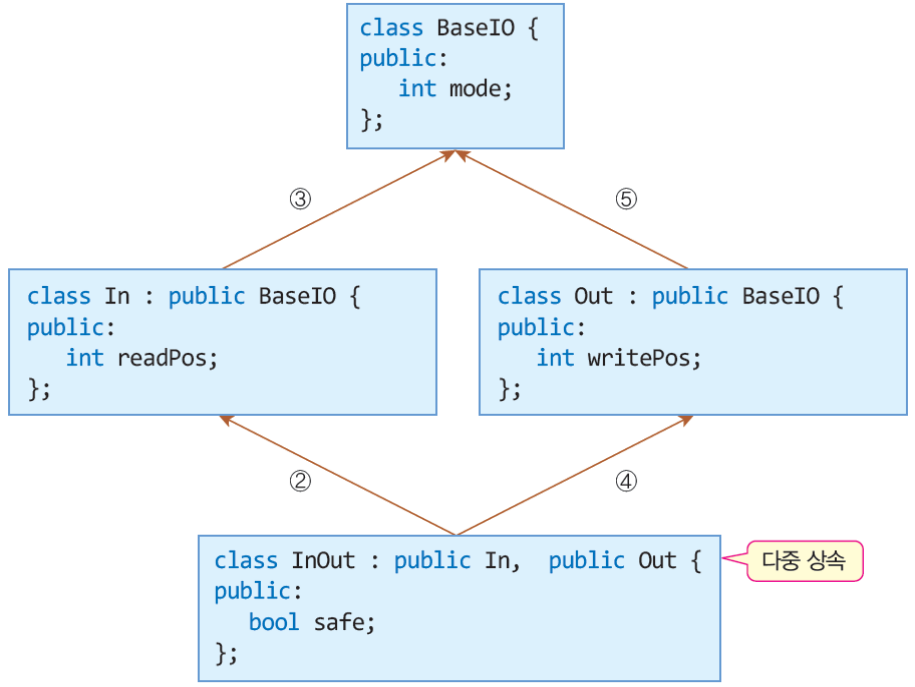
다중 상속 활용

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

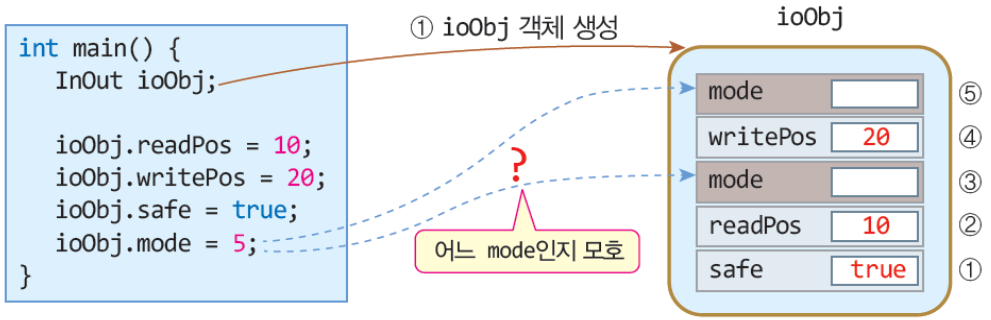
다중 상속의 문제점 : 기본 클래스 멤버 접근의 모호성

❖ 기본 클래스 멤버의 중복 상속

- Base의 멤버가 이중으로 객체에 삽입되는 문제점
- 멤버 변수 mode를 접근할 때, 어떤 mode에 접근해야 하는지 모호성이 발생



(a) 클래스 상속 관계



(b) ioObj 객체 생성 과정 및 객체 내부

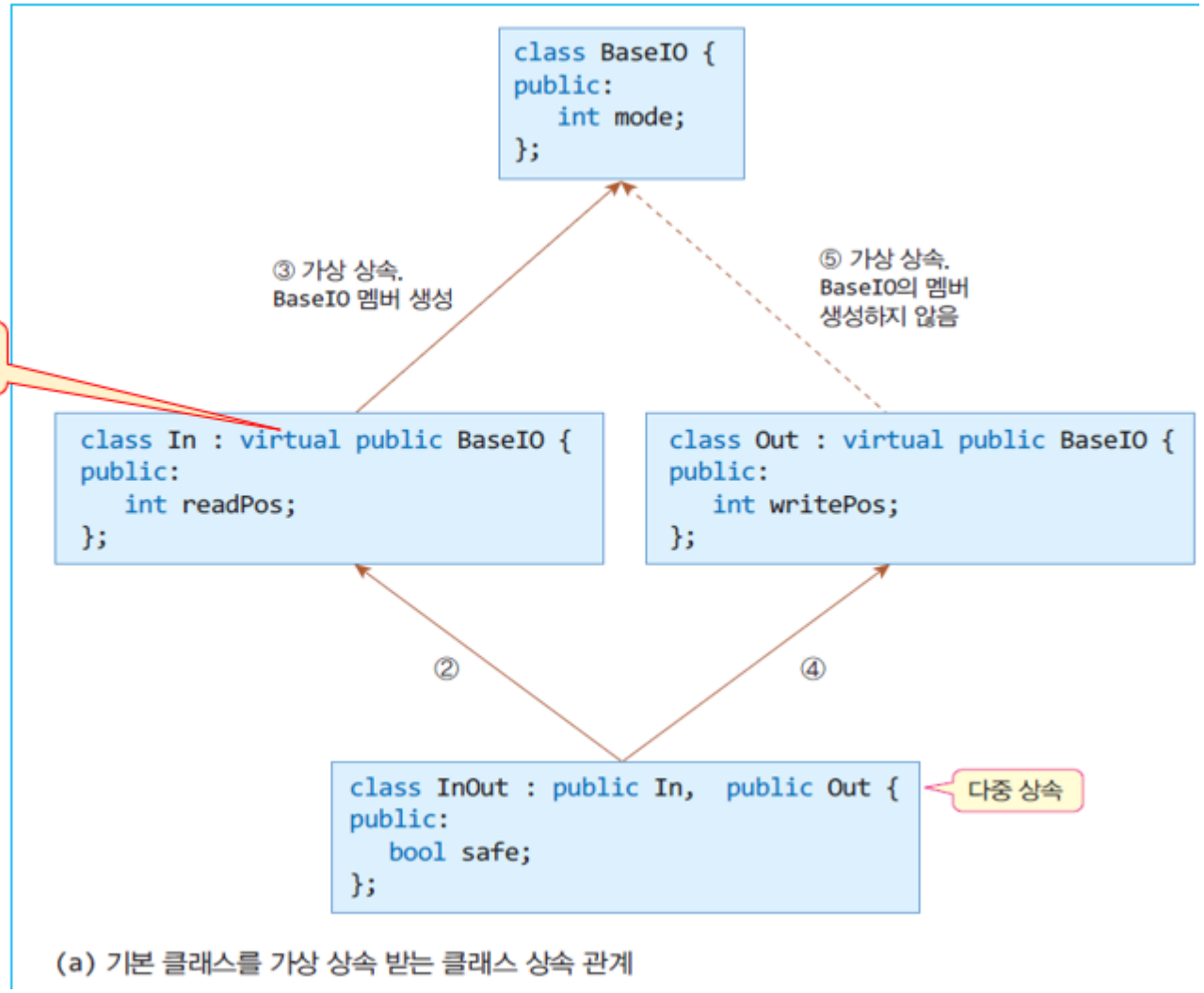
가상 상속

❖ 다중 상속으로 인한 **기본 클래스 멤버의 중복 상속 해결**

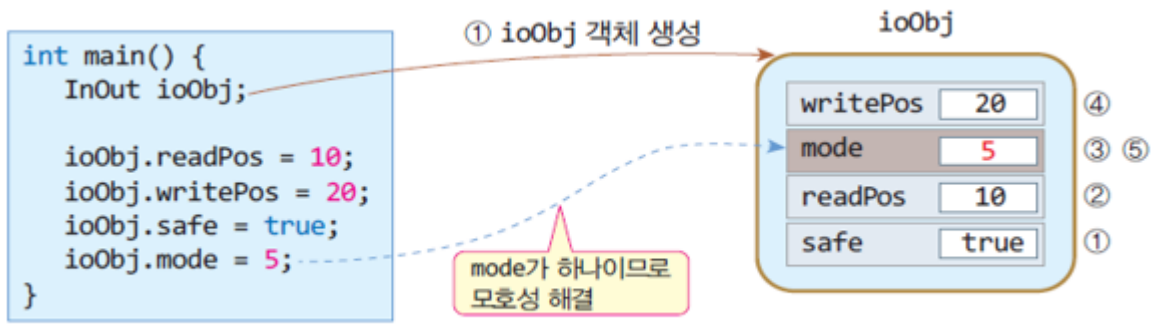
❖ 가상 상속

- 파생 클래스의 선언문에서 기본 클래스 앞에 **virtual**로 선언
- 파생 클래스의 객체가 생성될 때 기본 클래스의 멤버 공간을 한번만 할당. 이미 할당되어 있다면 그 공간을 공유
  - 기본 클래스의 멤버가 중복하여 생성되는 것을 방지

가상 상속으로 다중 상속의 모호성 해결



(a) 기본 클래스를 가상 상속 받는 클래스 상속 관계



(b) 가상 기본 클래스를 가진 경우, ioObj 객체 생성 과정 및 객체 내부

충돌의 가능성

❖2 개의 부모 클래스에 같은 이름의 멤버가 있는 경우

```
class UnderGradStudent {
public:
    void Warn();  // 학사 경고
};

class DormStudent {
public:
    void Warn();  // 벌점 부여
};

class UnderGrad_DormStudent : public UnderGradStudent, public DormStudent {
    // 두 클래스의 기능을 모두 상속
};

int main() {
    UnderGrad_DormStudent std;
    std.Warn();                 // ❌ 오류 발생: 어떤 Warn()인지 모름 (모호성)
    std.UnderGradStudent::Warn();  // ✅ OK: 어떤 Warn()을 호출할지 명시했기 때문
}
```

다음 수업

❖ 가상함수와 추상클래스
- 1_ 클래스 형 변환 규칙
- 2_ 함수재정의와 가상함수
- 3_ 추상클래스와 인터페이스 상속