



상속

학습 목표

- 상속 개념을 이해하고 문제 해결에 활용할 수 있다.
- 상속이 적용된 클래스의 객체 생성과 객체 포인터를 이해할 수 있다.
- 여러가지 상속 방법을 이해한다.
- 다중 상속과 가상 상속을 이해한다.
- 상속을 활용한 코드 재사용을 프로그램에 구현할 수 있다.
- 상속을 활용한 객체 생성과 접근을 이해할 수 있다.

학습 목차

- 상속 개념
- 클래스상속과 객체
- 상속과 객체 포인터
- 접근 지정자
- 상속과 생성자, 소멸자
- 상속의 종류
- 다중 상속
- 가상 상속

상속(Inheritance) 개념

- C++에서의 상속이란?
 - 객체가 생성될 때 자신의 멤버 뿐 아니라 부모 클래스의 멤버를 포함할 것을 지시
 - 기본 클래스의 속성과 기능을 파생 클래스에 물려주는 것
 - 기본 클래스(base class, superclass) : 상속 해 주는 클래스. 부모 클래스
 - 파생 클래스(derived class, subclass) : 상속 받는 클래스. 자식 클래스
 - 기본 클래스의 속성과 기능을 물려받고 자신만의 속성과 기능을 추가하여 작성
 - 여러 개의 클래스를 동시에 상속 받는 다중 상속 허용
 - 상속은 클래스를 선언할 때
 - 클래스 이름 뒤에 콜론(:)을 입력하고
 - 접근 제한자(public, protected, private)와 베이스 클래스의 이름을 붙여서 만듦

상속 관계를 이용한 클래스의 간결화 예

기능이 중복된 4 개의 클래스

class Student

말하기
먹기
걷기
잠자기
공부하기

class StudentWorker

말하기
먹기
걷기
잠자기
공부하기
일하기

class Researcher

말하기
먹기
걷기
잠자기
연구하기

class Professor

말하기
먹기
걷기
잠자기
연구하기
가르치기

상속 관계로 클래스의 간결화

class Person

말하기
먹기
걷기
잠자기

공통 기능을 Person
클래스로 작성

상속

class Student

공부하기

class Researcher

연구하기

상속

class StudentWorker

일하기

상속

class Professor

가르치기

상속의 목적 및 장점

- 간결한 클래스 작성
 - 기본 클래스의 기능을 물려받아 파생 클래스를 간결하게 작성
- 클래스 간의 계층적 분류 및 관리가 용이함.
 - 상속은 클래스들의 구조적 관계 파악에 용이함
- 클래스 재사용과 확장을 통한 소프트웨어 생산성 향상
 - 빠른 소프트웨어 생산
 - 기존에 작성한 클래스의 재사용 – 상속
 - 상속받아 새로운 기능을 확장
 - 앞으로 있을 상속에 대비한 클래스의 객체 지향적 설계가 필요

상속의 선언

```
class Student : public Person { // Person을 상속받는 Student 선언
    // Person 클래스의 멤버를 물려 받음
    //기본 클래스명 : Person
    //파생 클래스명: Student
    //상속 접근 지정 – private, protected 도 가능
    ....
};

class StudentWorker : public Student { // Student를 상속받는 StudentWorker 선언
    // Student가 물려받은 Person의 멤버도 함께 물려받음
    ....
};
```

상속 예

- Point 클래스를 상속받는 ColorPoint 클래스

// 2차원 평면에서 컬러 점을 표현하는 클래스 ColorPoint, Point를 상속받음

```
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(); // 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
}
```

// 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;
    }
};
```


파생 클래스의 객체 구성

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

Point p;

int x
 int y
 void set() {...}
 void showPoint() {...}

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

ColorPoint cp;

int x
 int y
 void set() {...}
 void showPoint() {...}

string color
 void setColor () {...}
 void showColorPoint() { ... }

파생 클래스의 객체는
기본 클래스의 멤버 포함

기본클래스 멤버

파생클래스 멤버

파생 클래스에서 기본 클래스 멤버 접근

파생클래스에서 기본
클래스 멤버 호출

x
y

```
void set(int x, int y) {  
    this->x= x; this->y=y;  
}
```

```
void showPoint() {  
    cout << x << y;  
}
```

Point 멤버

color

```
void setColor ( ) { ... }
```

```
void showColorPoint()  
{
```

```
    cout << color < ":";  
    showPoint();  
}
```

ColorPoint 멤버

ColorPoint cp 객체

외부에서 파생 클래스 객체에 대한 접근

x, y는 Point 클래스에 private이므로
set(), showPoint()에서만 접근 가능

기본 클래스
멤버 호출

```
ColorPoint cp;
```

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

main()

파생 클래스
멤버 호출

파생 클래스
멤버 호출

x 3
y 4

```
void set(int x, int y) {  
    this->x= x; this->y=y;  
}  
void showPoint() {  
    cout << x << y;  
}
```

color "Red"

```
void setColor (string color) {  
    this->color = color;  
}  
void showColorPoint() {  
    cout << color << ":";  
    showPoint();  
}
```

ColorPoint cp 객체

업 캐스팅(up-casting)

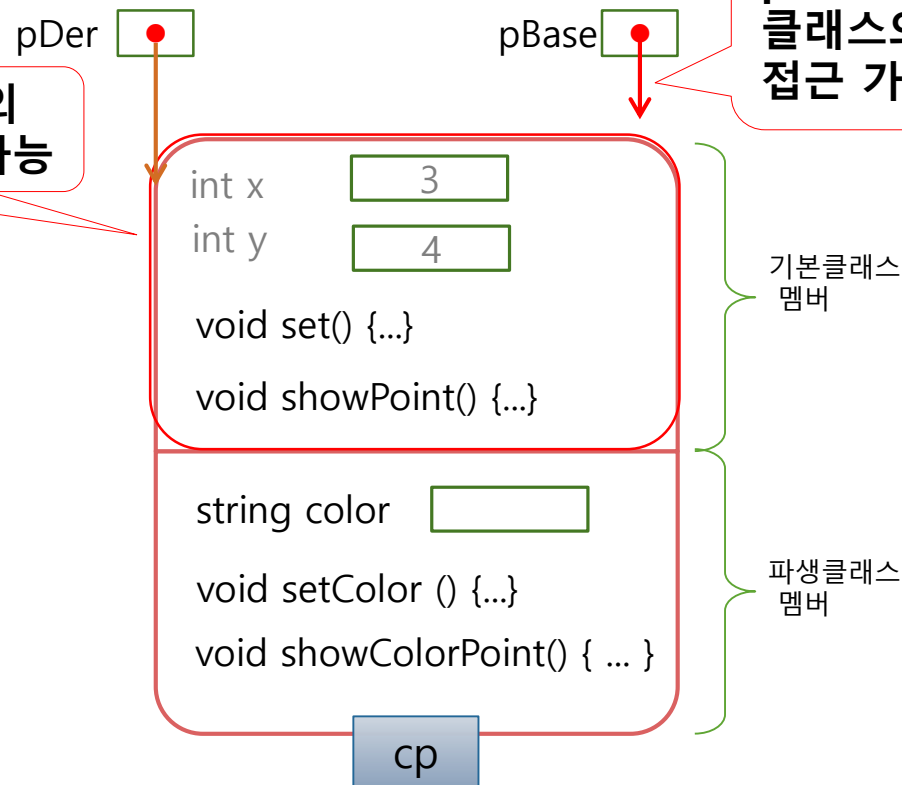
- 파생 클래스의 포인터가 기본 클래스의 포인터에 치환되는 것

```
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
}
```

pDer 포인터로 객체 cp의
모든 public 멤버 접근 가능

pBase 포인터로 기본
클래스의 public 멤버만
접근 가능



다운 캐스팅(down-casting)

- 기본 클래스의 포인터가 파생 클래스의 포인터에 치환되는 것

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

```
pBase->set(3,4);
pBase->showPoint();
```

//강제 타입 변환 반드시 필요

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

pDer 포인터로 객체 cp의
모든 public 멤버 접근 가능

pDer

pBase

pBase 포인터로 기본 클래스의
public 멤버만 접근 가능

int x 3
int y 4
void set() {...}
void showPoint() {...}

기본클래스 멤버

string color
void setColor () {...}
void showColorPoint() { ... }

파생클래스 멤버

cp

캐스팅(casting) 예

```
#include <iostream>
using namespace std;
#include <string>
class Point {
    int x, y; //한 점 (x,y) 좌표 값
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=" << x << ", y=" << y << endl;
}

class ColorPoint : public Point { //Point 클래스 상속
    string color; //점의 색 표현
public:
    void setColor(string color);
    void showColorPoint();
};
void ColorPoint::setColor(string color) {
    this->color = color;
}
void ColorPoint::showColorPoint() {
    cout << "color=" << color << endl;
    showPoint();
}
```

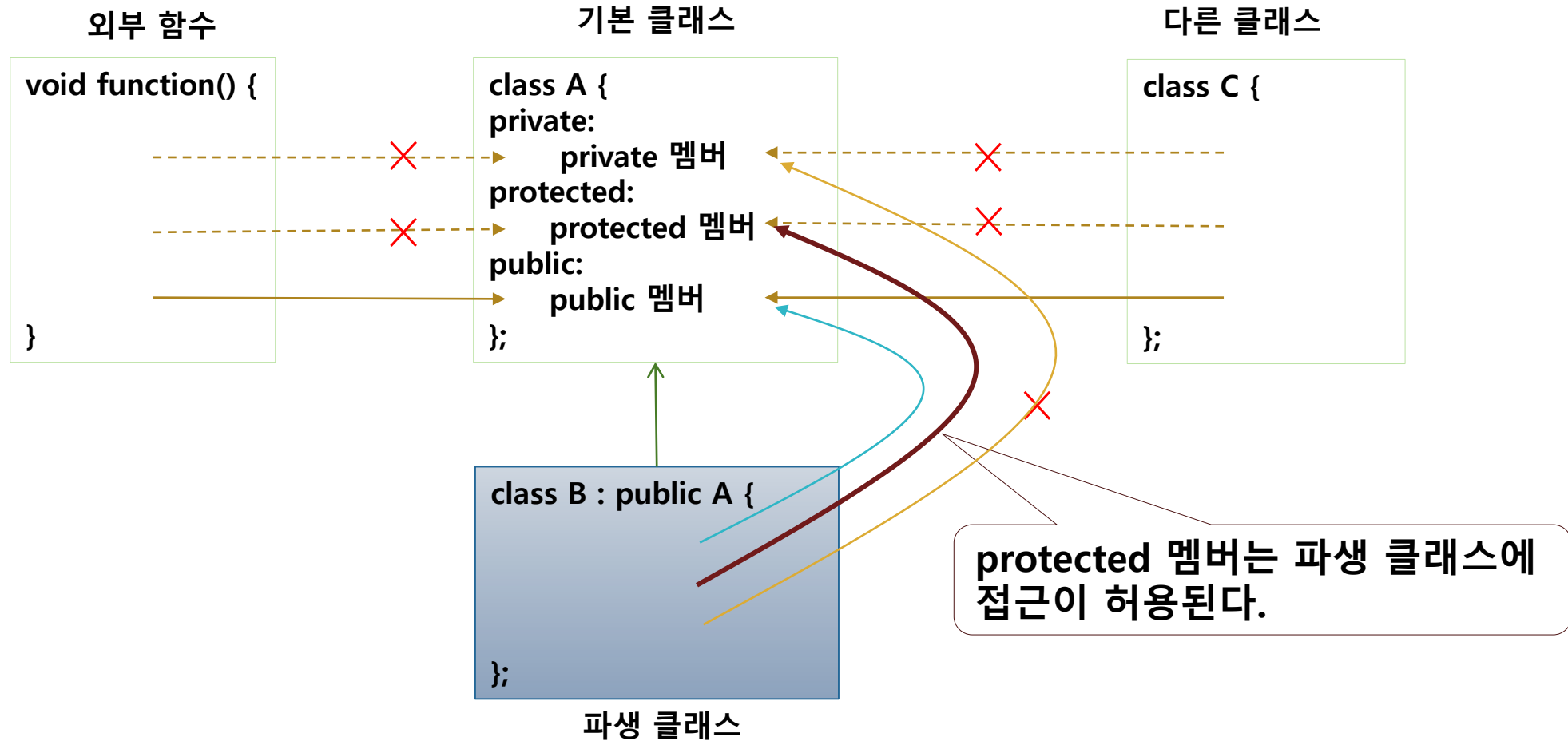
```
int main() {
    cout << "--AA-----" << endl;
    ColorPoint cp;
    Point *pBase = &cp; //업캐스팅
    pBase->set(3, 4);
    pBase->showPoint();
    cout << "--BB-----" << endl;
    ColorPoint *pDer; //다운캐스팅시 강제 타입 변환 반드시 필요
    pDer = (ColorPoint *)pBase; //다운캐스팅
    pDer->setColor("Red"); //정상 컴파일
    pDer->showColorPoint(); //정상 컴파일
    cout << "--CC-----" << endl;
    Point b;
    b.set(3, 4);
    b.showPoint();
    cout << "--DD-----" << endl;
    ColorPoint *pDer2;
    pDer2 = (ColorPoint *)&b; //다운캐스팅??
    pDer2->setColor("Red");
    pDer2->showColorPoint();
}
```

```
PS E:\lecture
--AA-----
x=3, y=4
--BB-----
color=Red
x=3, y=4
--CC-----
x=3, y=4
--DD-----
PS E:\lecture
```

접근 지정자

- private 멤버
 - 선언된 클래스 내에서만 접근 가능
 - 파생 클래스에서도 기본 클래스의 private 멤버에는 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
 - 파생 클래스가 아닌 다른 클래스나 외부 함수에서는 protected 멤버를 접근할 수 없다.

멤버의 접근 지정에 따른 접근성



protected 멤버에 대한 접근

```
class Point {
    protected: //or private
        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
}

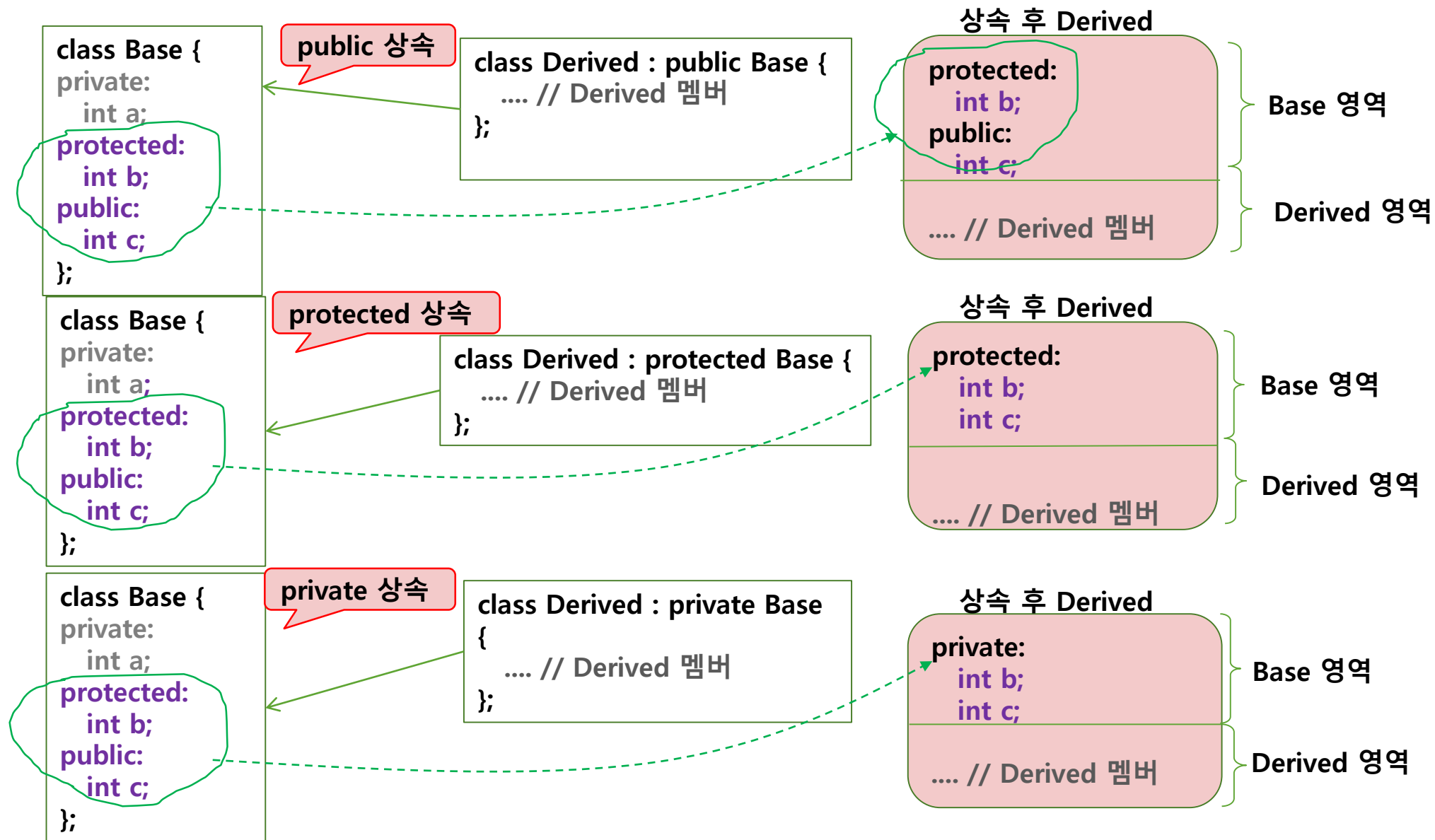
bool ColorPoint::equals(ColorPoint p) {
    if(x == p.x && y == p.y && color == p.color) // ①
        return true;
    else
        return false;
}

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
}
```

상속 시 접근 지정자의 속성

- 상속 지정
 - 상속 선언 시 public, private, protected의 3가지 중 하나 지정
 - 상속의 종류를 따로 지정하지 않으면 private 상속이 이루어짐
- 기본 클래스 멤버의 접근 속성을 어떻게 계승할지 지정
 - public : 기본 클래스의 protected, public 멤버 속성을 그대로 계승
 - protected : 기본 클래스의 protected, public 멤버를 protected로 계승
 - private : 기본 클래스의 protected, public 멤버를 private으로 계승

상속 시 멤버의 접근 지정 속성 변화



private 상속 예

다음에서 컴파일 오류가 발생하는 부분을 찾으세요.

```
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;           // error  
    x.setA(10);        // error  
    x.showA();         // error  
    x.b = 10;          // error  
    x.setB(10);        // error  
    x.showB();         //  
}
```

protected 상속 예

다음에서 컴파일 오류가 발생하는 부분을 찾으세요.

```
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;           // error  
    x.setA(10);        // error  
    x.showA();         // error  
    x.b = 10;          // error  
    x.setB(10);        // error  
    x.showB();         //  
}
```

상속이 중첩될 때 접근 지정 예

다음에서 컴파일 오류가 발생하는 부분을 찾으세요.

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

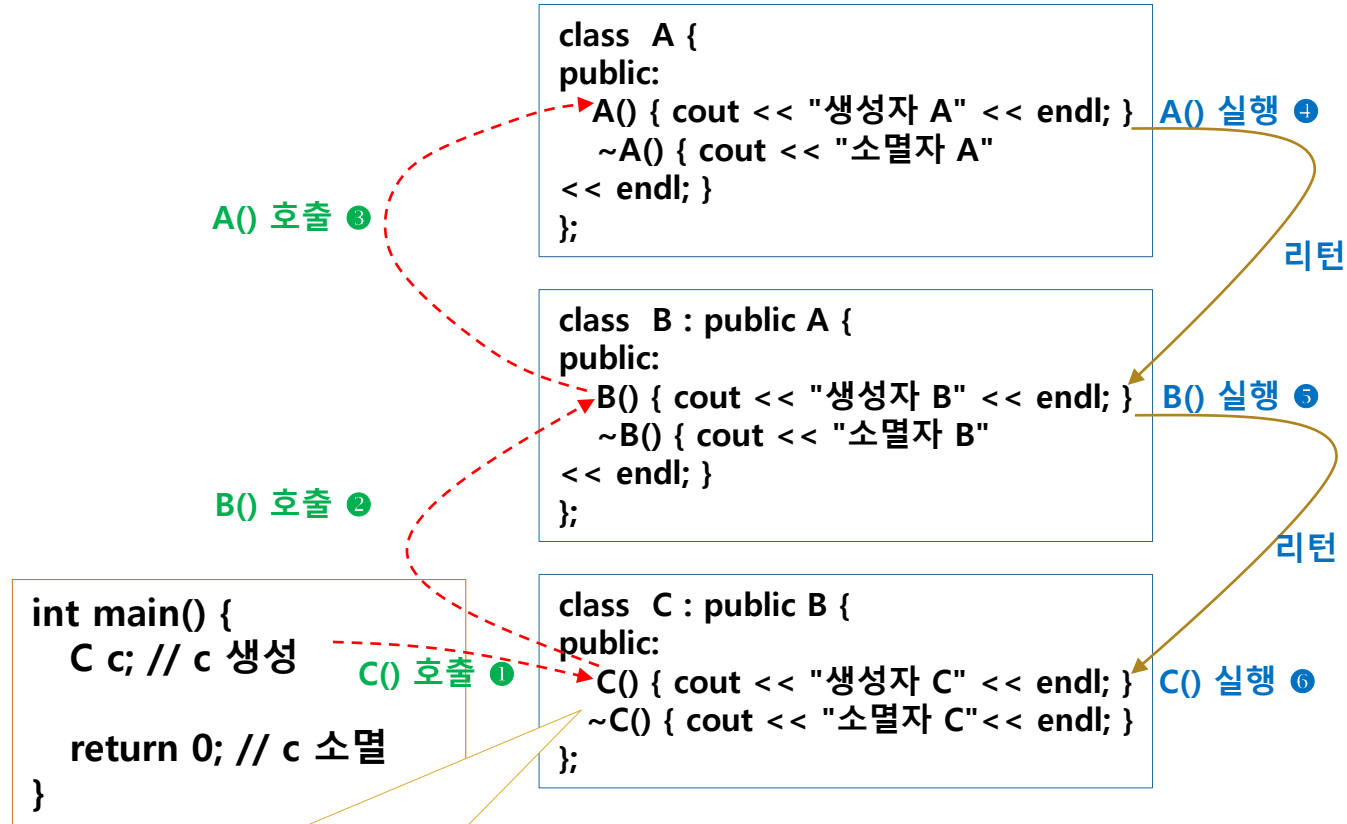
```
class Derived : private Base {
    int b;
protected:
    void setB(int b) { this->b = b; }
public:
    void showB() {
        setA(5);           // ①
        showA();           // ②
        cout << b;
    }
};
```

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

상속 관계에서 생성자와 소멸자의 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 실행.
- 소멸자 실행 순서
 - 파생 클래스의 객체가 소멸될 때
 - 파생 클래스의 소멸자가 먼저 실행되고,
 - 기본 클래스의 소멸자가 나중에 실행 됨.

생성자 호출 및 실행 순서



생성자 A
 생성자 B
 생성자 C
 소멸자 C
 소멸자 B
 소멸자 A

컴파일러는 C() 생성자 실행 코드를 만들 때, 생성자 B()를 호출하는 코드 삽입

default 생성자 호출 (1)

- 컴파일러에 의해 묵시적으로 기본 클래스의 생성자를 선택하는 경우

파생 클래스의 생성자에서 기본 클래스의 기본 생성자 호출

컴파일러는 묵시적으로 기본 클래스의 기본 생성자를 호출하도록 컴파일함

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

생성자 A
생성자 B

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

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

default 생성자 호출 (2)

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

컴파일러는 묵시적으로 기본 클래스의 기본 생성자를 호출하도록 컴파일 함.

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

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

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

생성자 A
매개변수생성자 B5

default 생성자 호출 (3)

- 컴파일러의 기본 생성자 호출 코드 삽입

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

컴파일러가 묵시적으로 삽입한 코드

컴파일러가 묵시적으로 삽입한 코드

default 생성자가 없는 경우

컴파일러가 B()에 대한
짝으로 A()를 찾을 수 없음.

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

```
class A {  
public:
```

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

컴파일 오류 발생 !!!

error C2512: 'A' : 사용할 수 있는
적절한 기본 생성자가 없습니다.

```
class B : public A {  
public:
```

```
    B() { // A() 호출하도록 컴파일 됨  
        cout << "생성자 B" << endl;  
    }  
};
```

명시적 생성자 호출

- 파생 클래스의 생성자에서 명시적으로 기본 클래스의 생성자 선택

파생 클래스의 생성자가
명시적으로 기본 클래스의
생성자를 선택 호출함

A(8) 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(x+3) {
        cout << "매개변수생성자 B" << x << endl;
    }
};
```

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

B(5) 호출

매개변수생성자 A8
매개변수생성자 B5

TV, WideTV, SmartTV 생성자 매개 변수 전달

```

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; }
};

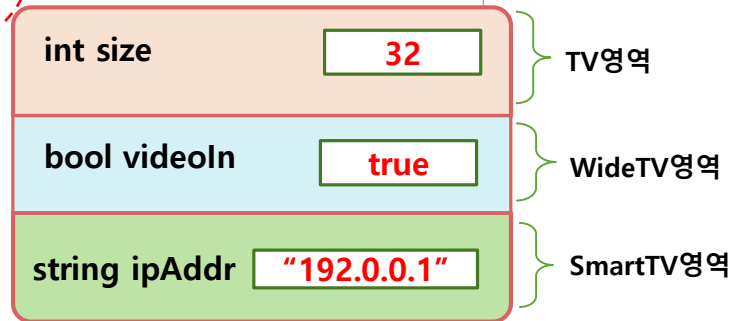
class SmartTV : public WideTV { //WideTV를 상속받는 SmartTV
    string ipAddr; //인터넷 주소
public:
    SmartTV(string ipAddr, int size) : WideTV(size, true) {
        this->ipAddr = ipAddr;
    }
    string getIpAddr() { return ipAddr; }
};

```

```

int main() {
    // 32 인치 크기에 "192.0.0.1"의 인터넷 주소를 가지는
    //스마트 TV 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;
}

```



다중 상속 선언 및 멤버 호출

다중 상속 활용

```
class MP3 {
public:
    void play();
    void stop();
};

class MobilePhone {
public:
    bool sendCall();
    bool receiveCall();
    bool sendSMS();
    bool receiveSMS();
};
```

다중 상속 활용

// 다중 상속 선언

```
class MusicPhone : public MP3, public MobilePhone {
public:
    void dial();
};
```

상속받고자 하는 기본 클래스 나열.

```
void MusicPhone::dial() {
    play();      //MP3::play() 호출
    sendCall();  //MobilePhone::sendCall() 호출
}
```

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

다중 상속 예

Adder와 Subtractor를 다중 상속받는 Calculator를 작성하세요.

```
#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; }
};
```

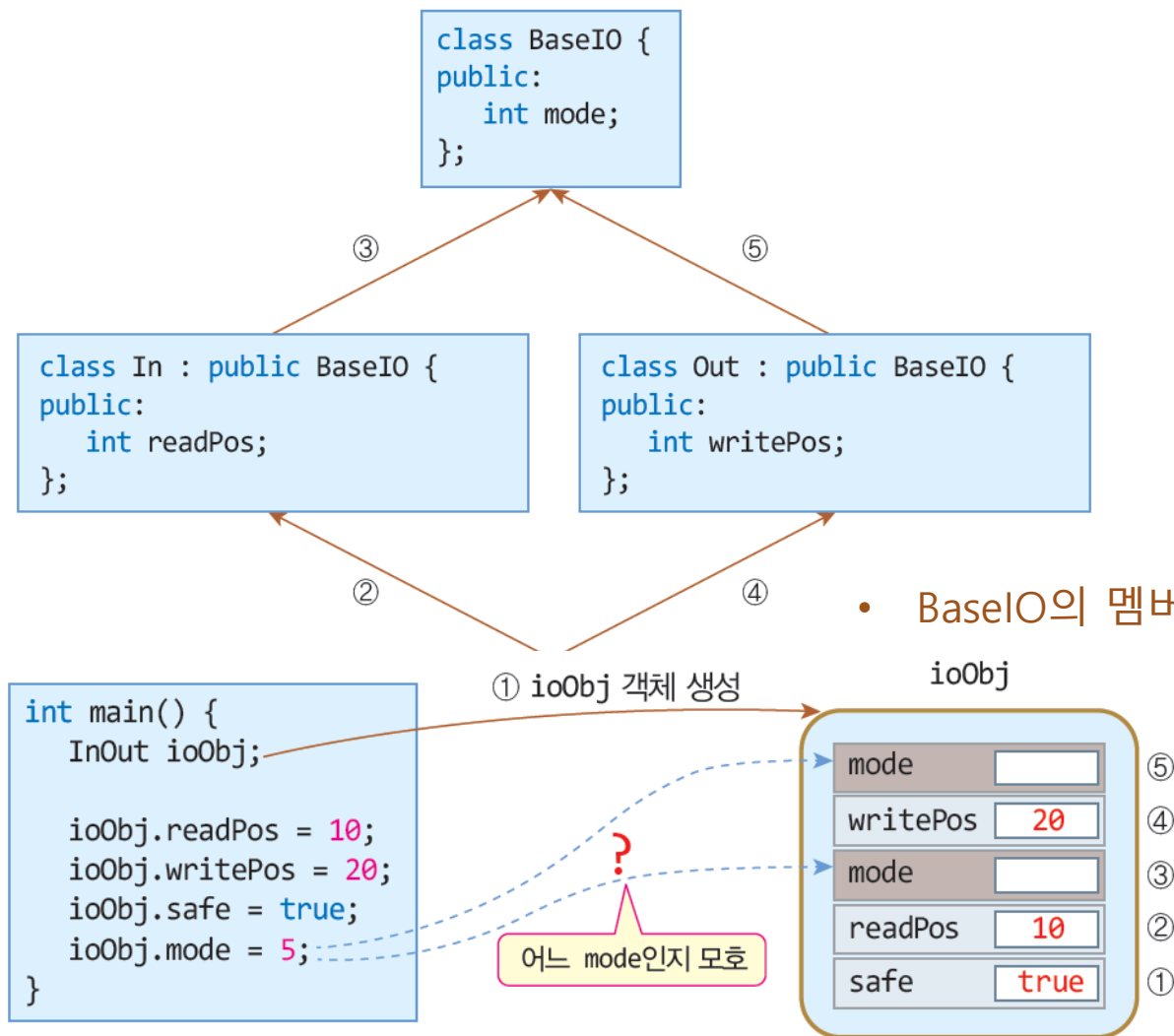
```
// 다중 상속
class Calculator : public Adder, public Subtractor {
public:
    int calc(char op, int a, int b);
};

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;
}
```

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


다중 상속의 문제점

- 기본 클래스 멤버의 중복 상속



- BaseIO의 멤버가 이중으로 객체에 삽입

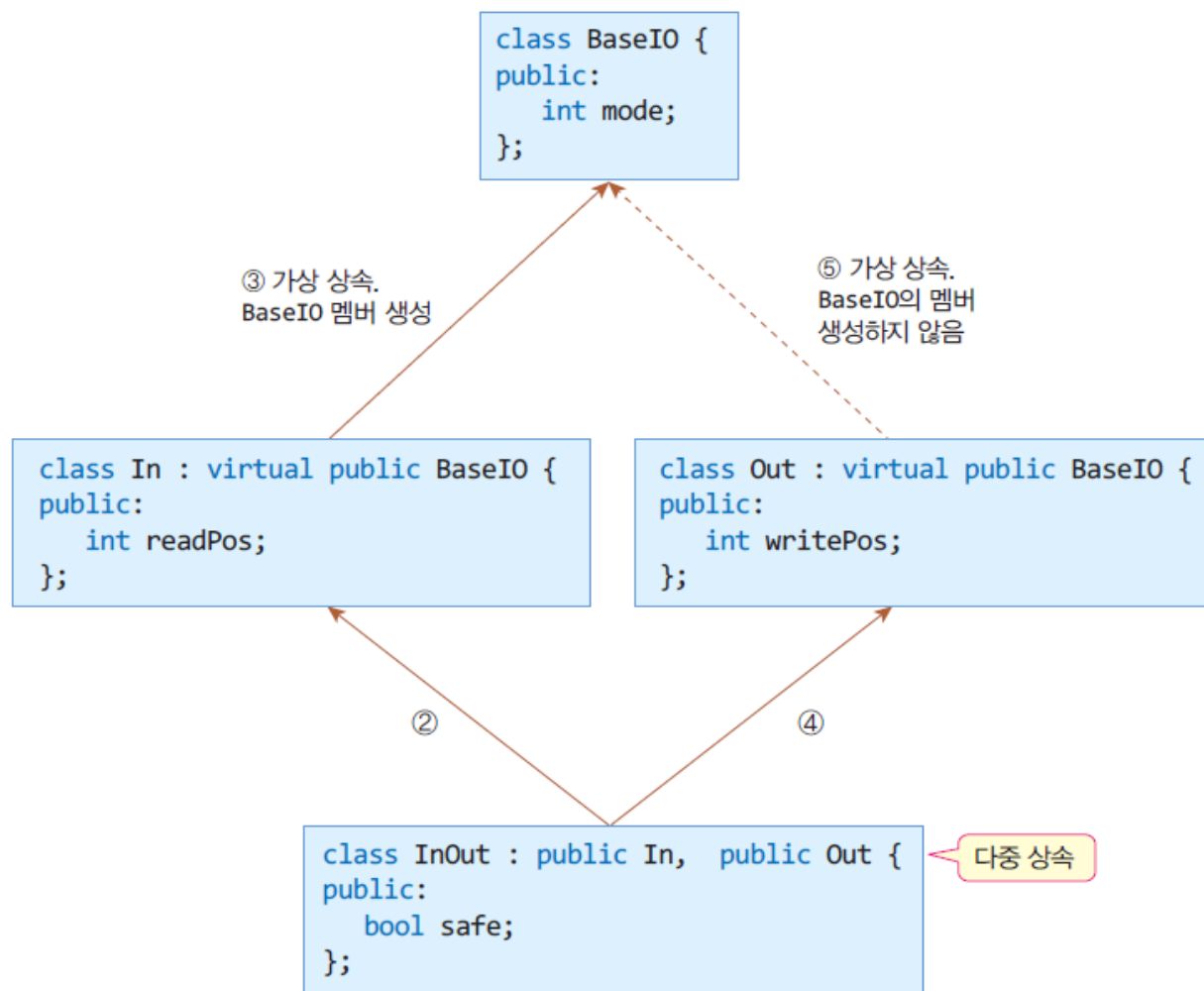
(b) ioObj 객체 생성 과정 및 객체 내부

가상 상속

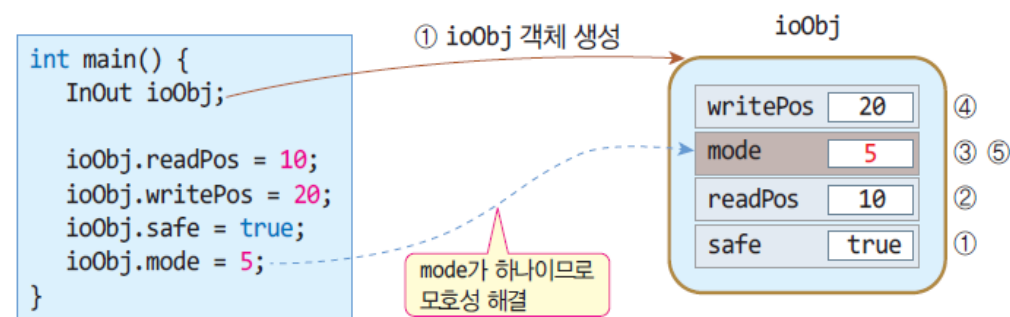
- 다중 상속으로 인한 기본 클래스 멤버의 중복 상속을 해결
- 가상 상속
 - 파생 클래스의 선언문에서 기본 클래스 앞에 **virtual**로 선언
 - 파생 클래스의 객체가 생성될 때 기본 클래스의 멤버는 오직 한 번만 생성
 - 기본 클래스의 멤버가 중복하여 생성되는 것을 방지

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

가상 상속으로 다중 상속의 모호성 해결



(a) 기본 클래스를 가상 상속 받는 클래스 상속 관계

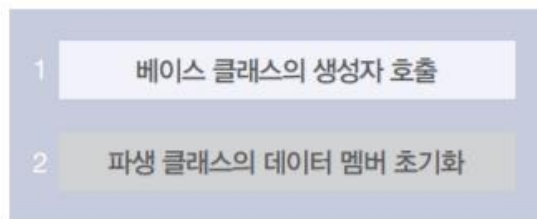


(b) 가상 기본 클래스를 가진 경우, ioObj 객체 생성 과정 및 객체 내부

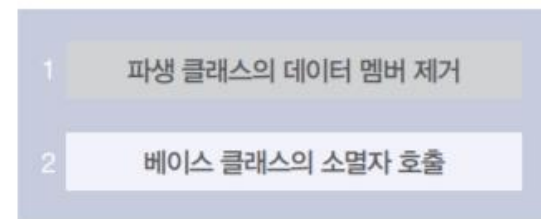
상속 되지 않는 멤버

- 기본 생성자, 매개변수가 있는 생성자, 복사 생성자, 소멸자, 대입 연산자

- 파생 클래스는 일반적으로 베이스 클래스보다 많은 데이터 멤버를 가지므로, 생성자와 소멸자는 상속되지 않음.
- 파생 클래스에서는 더 많은 데이터 멤버를 초기화 해야 하고, 그에 따라서 더 많은 데이터 멤버를 소멸시켜야 함.
- 그러나, 파생 클래스의 멤버 함수에서 베이스 클래스의 private 데이터 멤버에는 접근할 수 없으므로, 파생 클래스의 생성자에서 상속된 private 데이터 멤버는 접근 할 수 없어 초기화 불가능.
- 소멸자에서도 상속된 private 데이터 멤버를 소멸시킬 수 없음.
- 파생 클래스의 생성자에서 베이스 클래스의 생성자를 호출하고, 소멸자에서 베이스 클래스의 소멸자를 호출하는 형태로 해결할 수 있음.**



파생 클래스의 생성자



파생 클래스의 소멸자

상속 때의 생성자와 소멸자

학습 정리

- 상속의 선언
 - `class DerivedClassName : 접근지정자 BaseClassName { }`
 - 접근지정자는 `public`, `private`, `protected`의 3가지 중 하나, 상속의 종류를 따로 지정하지 않으면 `private` 상속
- 다중 상속 가능
 - 가상 상속으로 다중 상속의 모호성 해결 `class DerivedClassName : virtual 접근지정자 BaseClassName { }`
- 업 캐스팅(up-casting) : 기본 클래스의 포인터 = 파생 클래스의 포인터;
- 다운 캐스팅(down-casting) : 파생 클래스의 포인터 = 기본 클래스의 포인터;
- 상속 관계에서 생성자와 소멸자의 실행 : 생성(기본->파생), 소멸(파생->기본)
- 상속 되지 않는 멤버 : 기본 생성자, 매개변수가 있는 생성자, 복사 생성자, 소멸자, 대입 연산자

Q & A

- “상속”에 대한 학습이 모두 끝났습니다.
- 새로운 내용이 많았습니다. 모든 내용을 이해 하셨나요?
- 아직 이해가 안되는 내용이 있다면 다시 한번 복습하시기 바랍니다.
- 질문은 한림 SmartLEAD 쪽지 또는 e-mail 또는 전화상담을 이용하시기 바랍니다.



- cpp_09_상속_ex.pdf 에 확인 학습 문제들을 담았습니다.
- 이론 학습을 완료한 후 확인 학습 문제들로 학습 내용을 점검 하시기 바랍니다.
- 퀴즈와 과제가 출제되었습니다. 마감시간에 늦지 않도록 주의해 주세요.
- 수고하셨습니다.^^