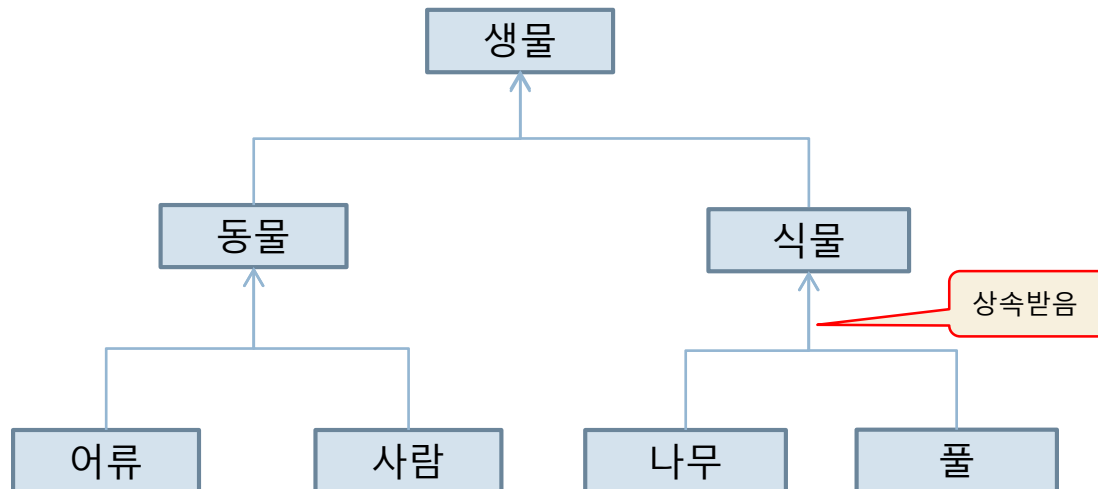


1

# Ch11 상속

# 유전적 상속과 객체 지향 상속

2



유전적 상속과 관계된  
생물 분류

# C++에서의 상속(Inheritance)

3

- C++에서의 상속이란?
  - ▣ 클래스 사이에서 상속관계 정의
    - 객체 사이에는 상속 관계 없음
  - ▣ 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 자신 만의 속성과 기능을 추가하여 작성
  - ▣ 기본 클래스에서 파생 클래스로 갈수록 클래스의 개념이 구체화
  - ▣ 다중 상속을 통한 클래스의 재활용성 높임

# 상속의 표현

4



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

5

## 1. 간결한 클래스 작성

- 기본 클래스의 기능을 물려받아 파생 클래스를 간결하게 작성

## 2. 클래스 간의 계층적 분류 및 관리의 용이함

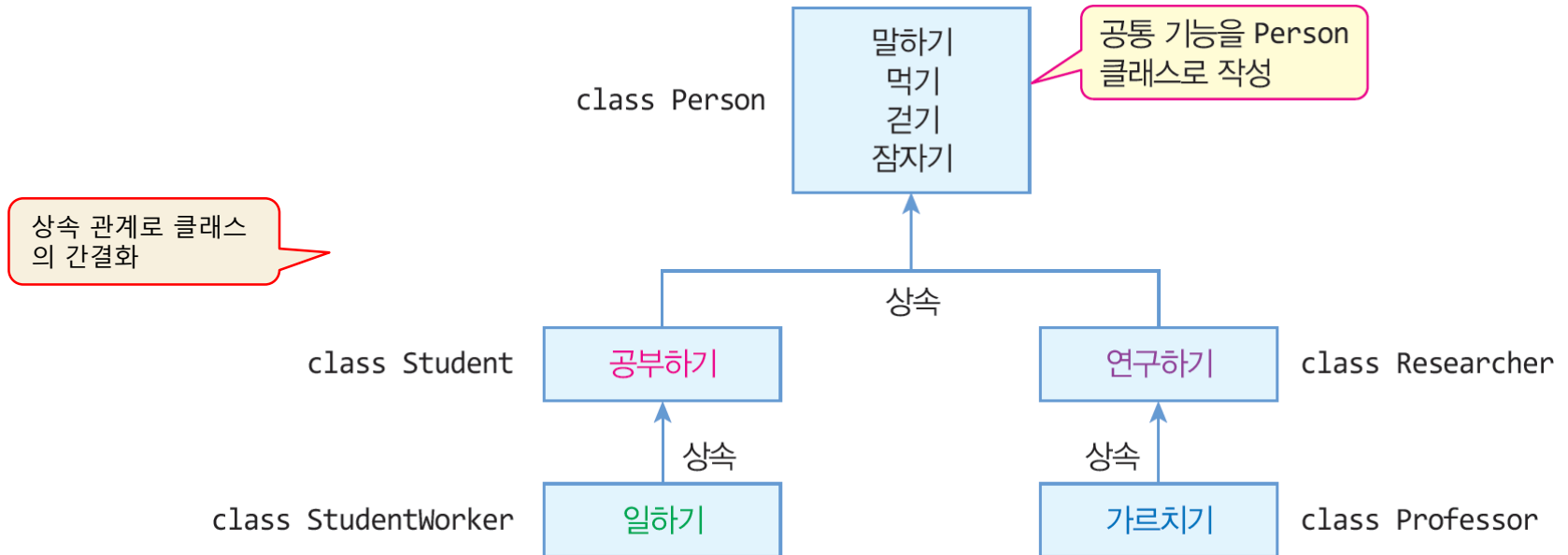
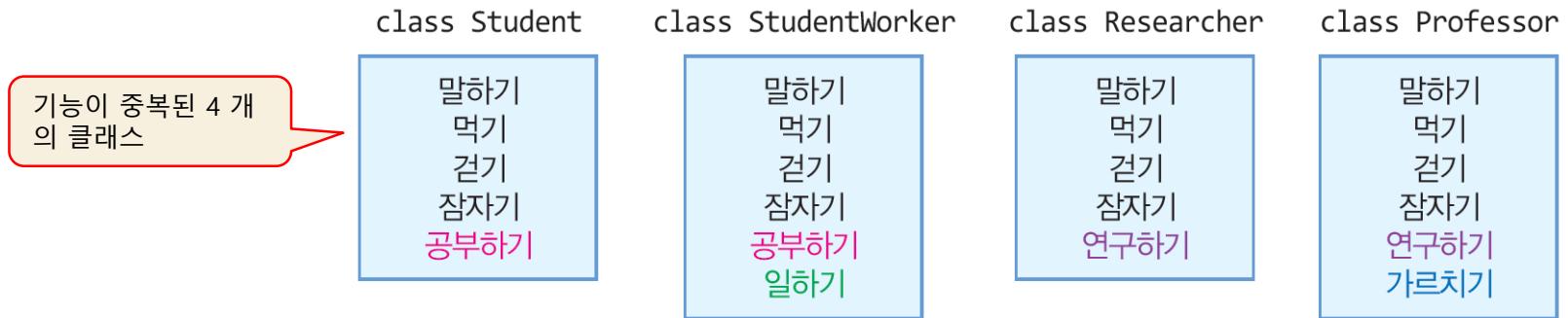
- 상속은 클래스들의 구조적 관계 파악 용이

## 3. 클래스 재사용과 확장을 통한 소프트웨어 생산성 향상

- 빠른 소프트웨어 생산 필요
- 기존에 작성한 클래스의 재사용 – 상속
  - 상속받아 새로운 기능을 확장
- 앞으로 있을 상속에 대비한 클래스의 객체 지향적 설계 필요

# 상속 관계로 클래스의 간결화 사례

6



# 상속 선언

7

## □ 형식

파생클래스명

상속 접근 지정.  
private, protected  
도 가능

기본클래스명

```
class Derived : public Base {  
  
};
```

파생클래스명

상속 접근 지정.  
private, protected  
도 가능

기본클래스명

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

- Student 클래스는 Person 클래스의 멤버를 물려받는다.

# 상속 선언

8

## □ 형식

상속 접근 지정.  
private, protected  
도 가능

파생클래스명

기본클래스명

```
class Derived : public Base {  
};
```

상속 형식	기반 클래스의 public 멤버	기반 클래스의 protected 멤버	기반 클래스의 private 멤버
public Base	public 으로 유지	protected 으로 유지	접근 불가
protected Base	protected 로 변경	protected 으로 유지	접근 불가
private Base	private 로 변경	private 로 변경	접근 불가



# 예제 8-1 Point 클래스를 상속받는 ColorPoint 클래스 만들기

9

```
#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);
    void showPoint();
};

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

void Point::showPoint() {
    cout << x << "," << y << endl;
}
```

```
// 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
}
```

Red:3,4

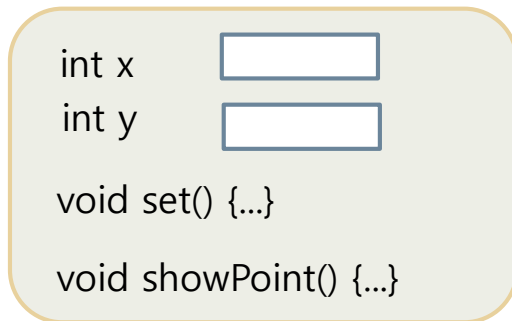
# 파생 클래스의 객체 구성

10

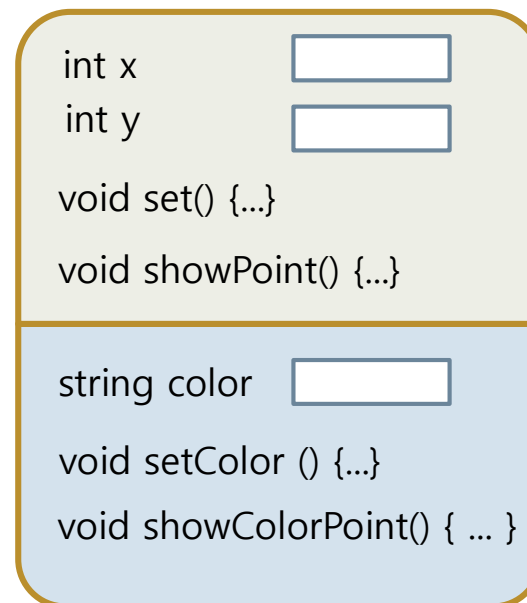
```
class Point {  
    int x, y; // 한 점 (x,y) 좌표 값  
public:  
    void set(int x, int y);  
    void showPoint();  
};
```

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

Point p;



ColorPoint cp;



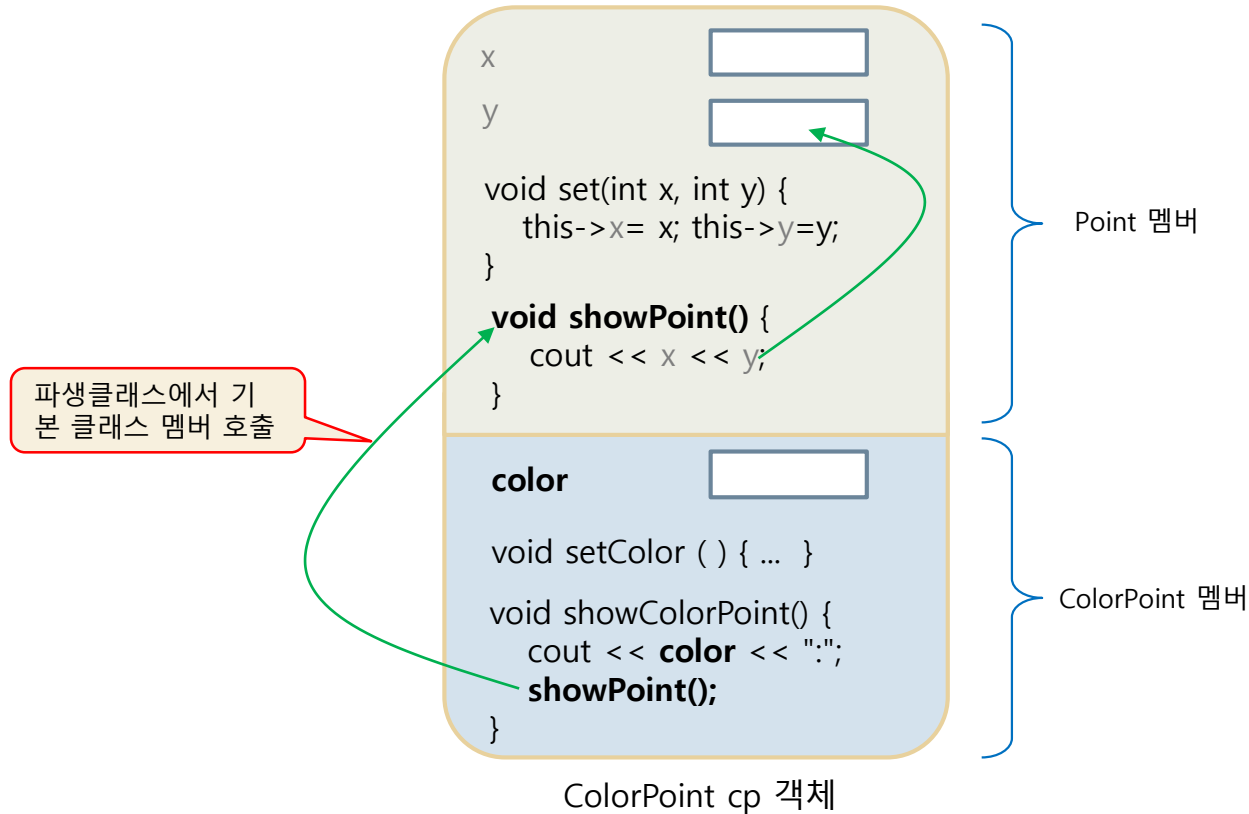
파생 클래스의 객체는 기본 클래스의 멤버 포함

기본클래스 멤버

파생클래스 멤버

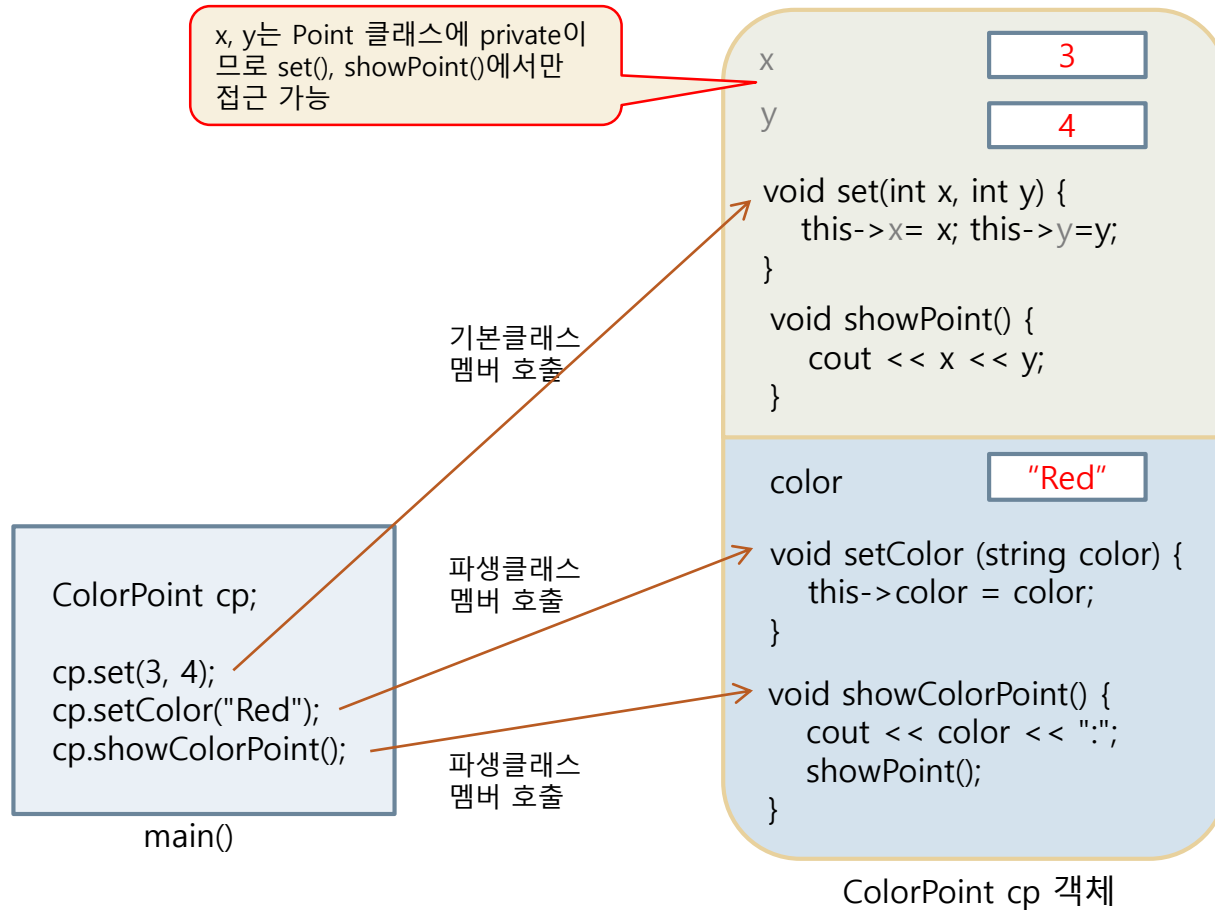
# 파생 클래스에서 기본 클래스 멤버 접근

11



# 외부에서 파생 클래스 객체에 대한 접근

12



# 상속과 객체 포인터 - 업 캐스팅

13

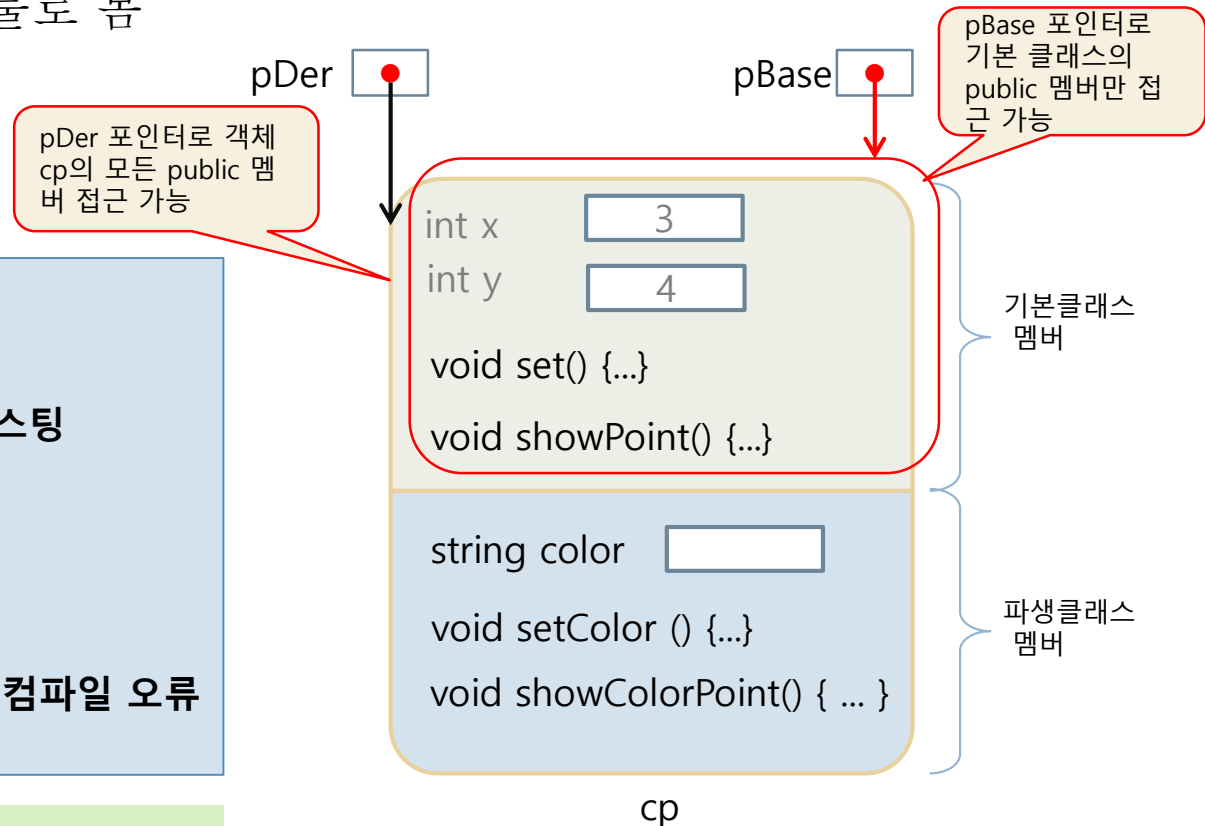
## □ 업 캐스팅(up-casting)

- 파생 클래스 포인터가 기본 클래스 포인터에 치환되는 것

- 예) 사람을 동물로 봄

```
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)



# 업 캐스팅

14



생물을 가리키는 손가락으로  
어류, 포유류, 사람, 식물 등  
생물의 속성을 상속받은 객체  
들을 가리키는 것은 자연스럽  
습니다. 이것이 업 캐스팅의  
개념입니다.

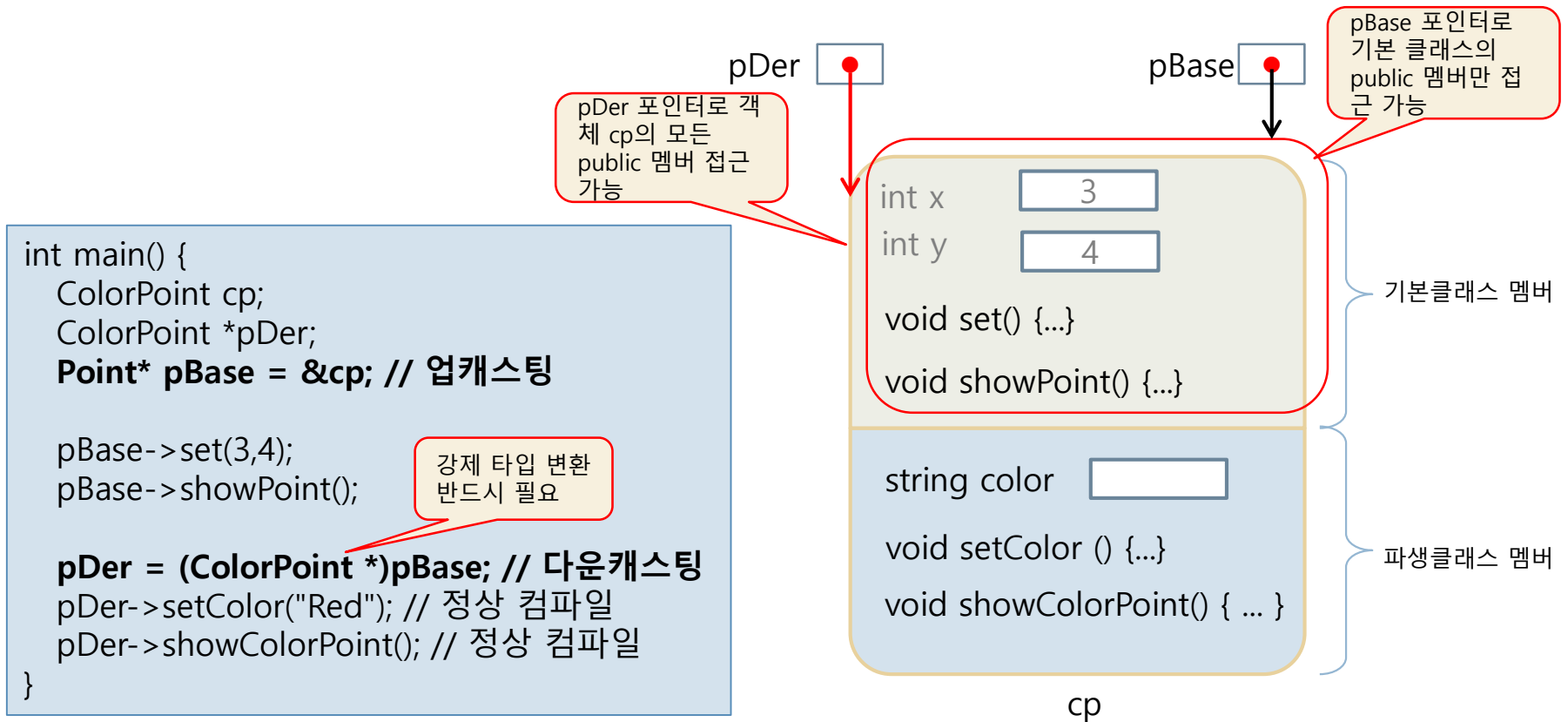


생물을 가리키는 손가락  
으로 컵을 가리키면 오류

# 상속과 객체 포인터 - 다운 캐스팅

15

- 다운 캐스팅(down-casting)
  - ▣ 기본 클래스의 포인터가 파생 클래스의 포인터에 치환되는 것



(3,4)  
Red(3,4)

# protected 접근 지정

16

## □ 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

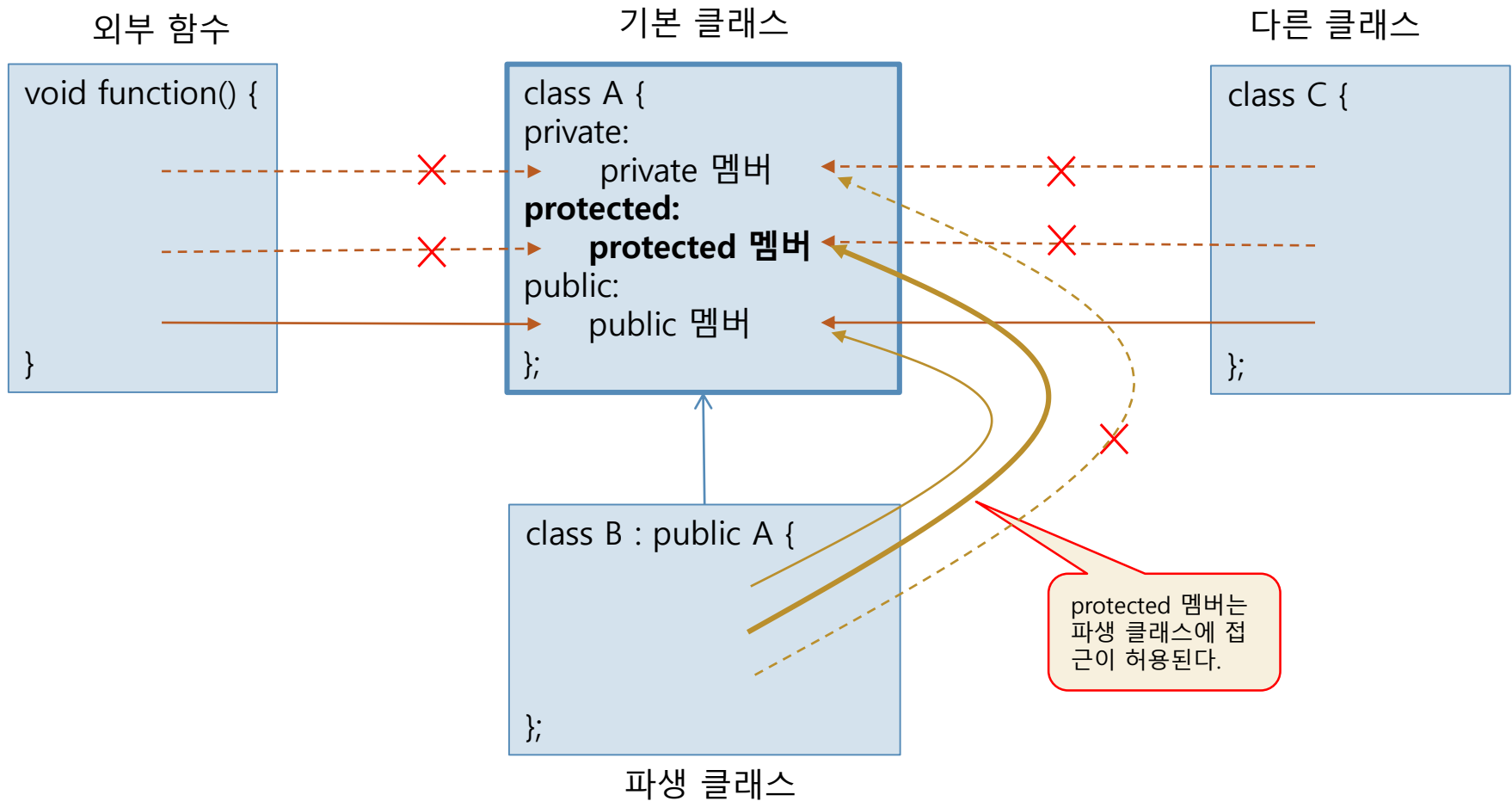
### ▣ protected 멤버

- 선언된 클래스에서 접근 가능
- 파생 클래스에서만 접근 허용
  - 파생 클래스가 아닌 다른 클래스나 외부 함수에서는 *protected* 멤버를 접근할 수 없다.



# 멤버의 접근 지정에 따른 접근성

17



## 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);
    p.x = 5;
    p.y = 5;
    p.showPoint();

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

// ②  
// ③ 오류  
// ④ 오류

// ⑤ 오류  
// ⑥ 오류

# 상속 관계의 생성자와 소멸자 실행

19

## □ 질문 1

- ▣ 파생 클래스의 객체가 생성될 때 파생 클래스의 생성자와 기본 클래스의 생성자가 모두 실행되는가? 아니면 파생 클래스의 생성자만 실행되는가?

■ 답 - 둘 다 실행된다.

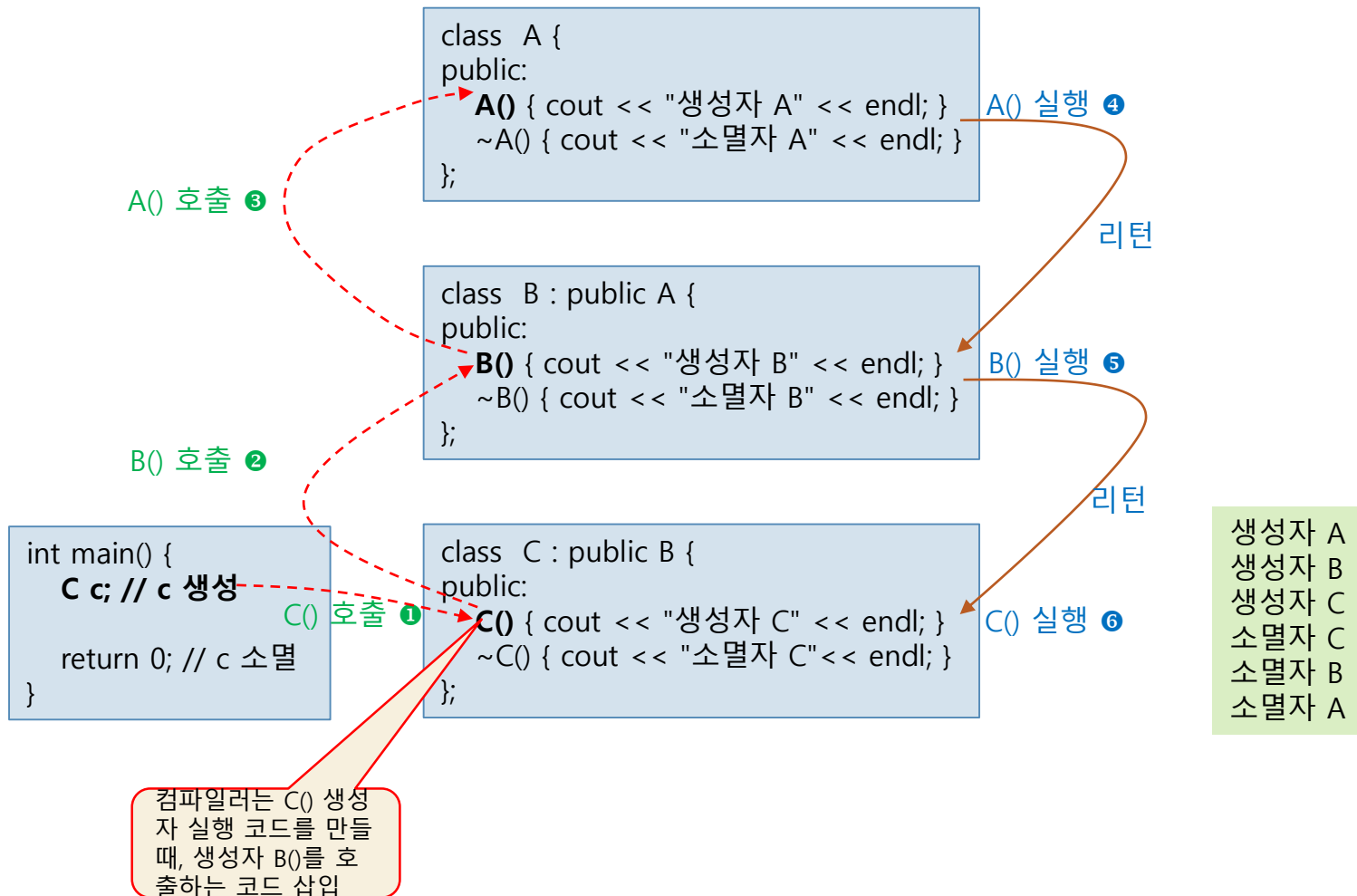
## □ 질문 2

- ▣ 파생 클래스의 생성자와 기본 클래스의 생성자 중 어떤 생성자가 먼저 실행되는가?

■ 답 - 기본 클래스의 생성자가 먼저 실행된 후 파생 클래스의 생성자가 실행된다.

# 생성자 호출 관계 및 실행 순서

20



# 소멸자의 실행 순서

21

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

# 컴파일러에 의해 묵시적으로 기본 클래스의 생성자를 선택하는 경우

22

파생 클래스의 생성자에서 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;  
    }  
};
```

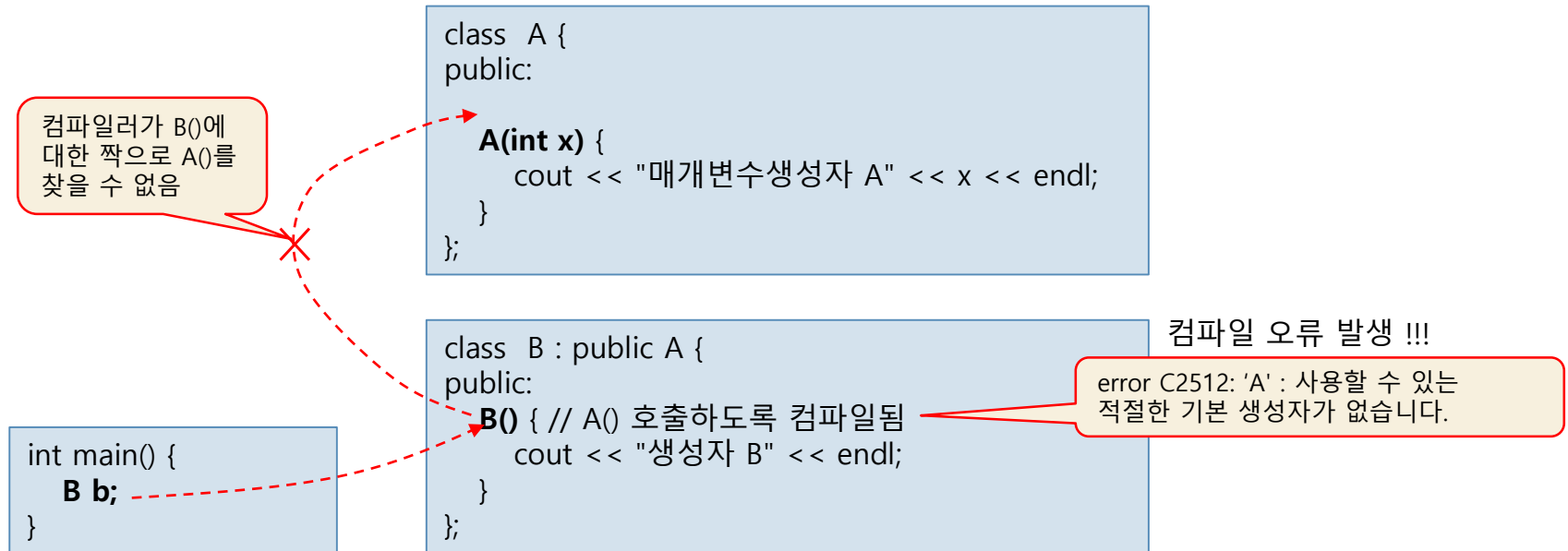
```
class B : public A {  
public:  
    ▶ B() { // A() 호출하도록 컴파일됨  
        cout << "생성자 B" << endl;  
    }  
};
```

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

생성자 A  
생성자 B

# 기본 클래스에 기본 생성자가 없는 경우

23



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

24

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



# 파생 클래스의 생성자에서 명시적으로 기본 클래스의 생성자 선택

25

파생 클래스의 생성자가 명시적으로 기본 클래스의 생성자를 선택 호출함

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

A(8) 호출

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

B(5) 호출

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

매개변수생성자 A8  
매개변수생성자 B5

# 컴파일러의 기본 생성자 호출 코드 삽입

26

```
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

# 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();
    Tv(int size);
    int getSize();
};
```

```
Tv::Tv() {
    size = 20;
}
```

```
Tv::Tv(int size) {
    this->size = size;
}
```

```
int Tv::getSize() {
    return size;
}
```

```
class WideTV : public Tv { // Tv를 상속받는 WideTV
```

```
    bool videoIn;
public:
    WideTV(int size, bool videoIn);
    bool getVideoIn();
};
```

```
WideTV::WideTV(int size, bool videoIn) : Tv(size) {
    this->videoIn = videoIn;
}
```

```
bool WideTV::getVideoIn() {
    return videoIn;
}
```

```
int main() {
    // 32 인치 크기 wide Tv 객체 생성
    WideTV htv(32,true);
    cout << "size=" << htv.getSize() << endl;
    cout << "videoIn=" << boolalpha << htv.getVideoIn() << endl;
}
```

32

32

true

size=32  
videoIn=true

int size

32

TV영역

bool videoIn

true

WideTV영역

htv