객체지향언어

9장 상속

학습목표

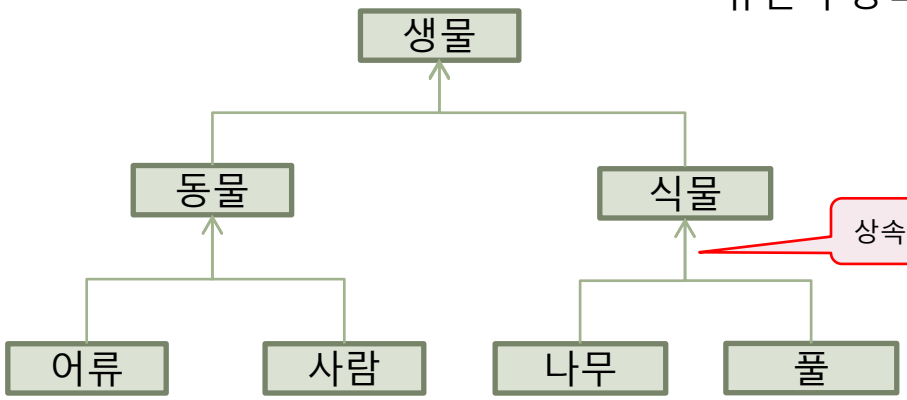
- 클래스 간의 상속의 개념을 이해하고, 상속을 사용할 수 있다.
- 클래스 간의 상속에서 멤버함수의 재정의를 설명하고, 사용할 수 있다.
- 부모-자식 클래스간의 형 변환을 설명할 수 있다.

상속의 개념

- **상속(Inheritance) 이란?**
  - 상속이란 부모로부터 무엇인가를 물려받는 것으로 상속을 받음으로 인해 남들 보다 쉽게 안정된 생활을 이룰 수가 있음

유산 상속

유전적 상속 : 객체 지향 상속



유전적 상속과 관계된 생물 분류

클래스간의 상속

C++에서의 상속(inheritance)

- C++에서 클래스 사이에 무엇인가를 물려주고 물려받는 것을 클래스의 상속이라고 하며, **클래스 사이에서 상속관계를 정의**하는 것을 의미함
  - 객체 사이에는 상속 관계 없음
- 기본(부모) 클래스의 속성과 기능을 파생(자식) 클래스에 물려주는 것으로 파생 클래스로 갈수록 클래스의 개념이 구체화됨
  - 기본(부모) 클래스(base class) - 상속해주는 클래스. 부모 클래스
  - 파생(자식) 클래스(derived class) – 상속받는 클래스. 자식 클래스
  - 파생(자식) 클래스는 기본(부모) 클래스의 속성과 기능을 물려받고 **자신 만의 속성과 기능을 추가하여 작성**
- 장점
  - 이미 정의된 클래스를 상속받아 새로운 클래스를 만든다면 **검증된 프로그램을 쉽게 작성할 수 있음**
  - 기본(부모) 클래스의 멤버 변수와 메서드(함수)를 그대로 사용할 수 있으므로 **코드를 재활용할 수 있고 코드의 중복을 줄일 수 있음**

클래스간의 상속

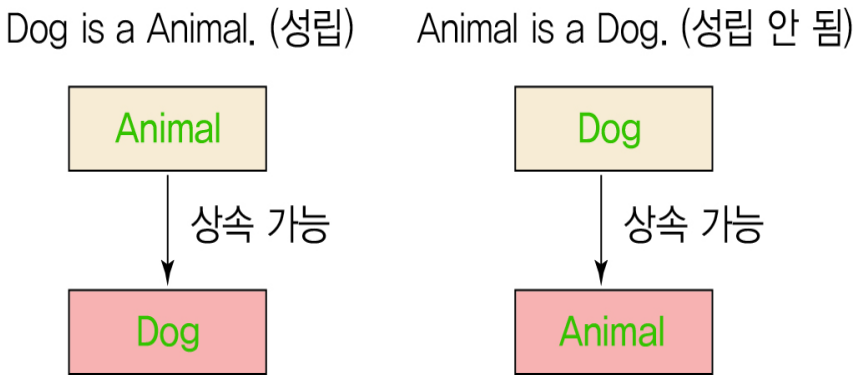
■ is-a 관계

- 두 개 사이에 is-a 관계가 성립하면 상속 관계

예) 개는 동물이다(Dog is a Animal.)인 관계는 is-a 관계

이런 경우에 Animal 클래스로부터 Dog 클래스가 상속될 수 있다. 이 때 Animal과 같이 일반적인 특징을 가진 것이 기본 클래스이고, 기본 클래스에 구체적인 특징을 추가한 Dog과 같은 것이 파생 클래스이다.

- is-a 관계는 **한 방향으로만 성립**한다.
  - 동물은 개다(Animal is a Dog.)의 관계는 성립하지 않는다.
  - Dog 클래스로부터 Animal 클래스가 상속될 수 없다.



클래스간의 상속

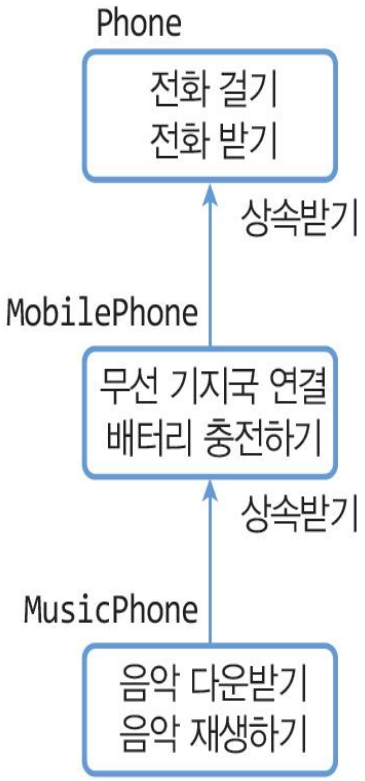
has-a 관계

- 두 개 사이에 has-a 관계가 성립하면 포함 관계
- has-a 관계는 **상속이 될 수 없고 클래스의 멤버 변수로 포함**될 수 있음

예) 개는 꼬리이다(Tail is a Dog.)의 관계는 성립하지 않고, 개는 꼬리를 가지고 있다(Dog has a Tail.)의 관계는 성립된다. 즉, Tail은 Dog 으로부터 상속받을 수 있는 것이 아니고, Tail 은 Dog 클래스의 멤버가 될 수 있다.

클래스간의 상속

■ 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

클래스간의 상속

■ 상속관계를 클래스로 설계할 때

- 멤버로 포함되어야 할지, 혹은 상속으로 되어야 할지에 따라 is-a관계와 has-a 관계를 고려하여 상속여부를 결정하여 설계에 반영할 수 있음

■ 상속관계를 설계하는 절차

- 1단계 : 먼저 기본(부모) 클래스와 파생(자식) 클래스 **모두의 공통적인 변수와 함수를 멤버로 하여 기본 클래스를 정의**
- 2단계 : 자식 클래스는 부모 클래스의 특성을 상속받고 **자신에게만 속하는 멤버를 추가하여 설계**

[예시] 동물의 기본 속성을 갖는 animal 클래스를 활용한 상속관계 설계

animal이라는 기본 클래스를 먼저 만들고, 그 다음에 animal 클래스에서 상속받아 tiger, lion, dog과 같은 파생 클래스를 만들어 각각에 고유한 특성을 추가한다.

클래스간의 상속

■ 상속이 사용 가능한 경우

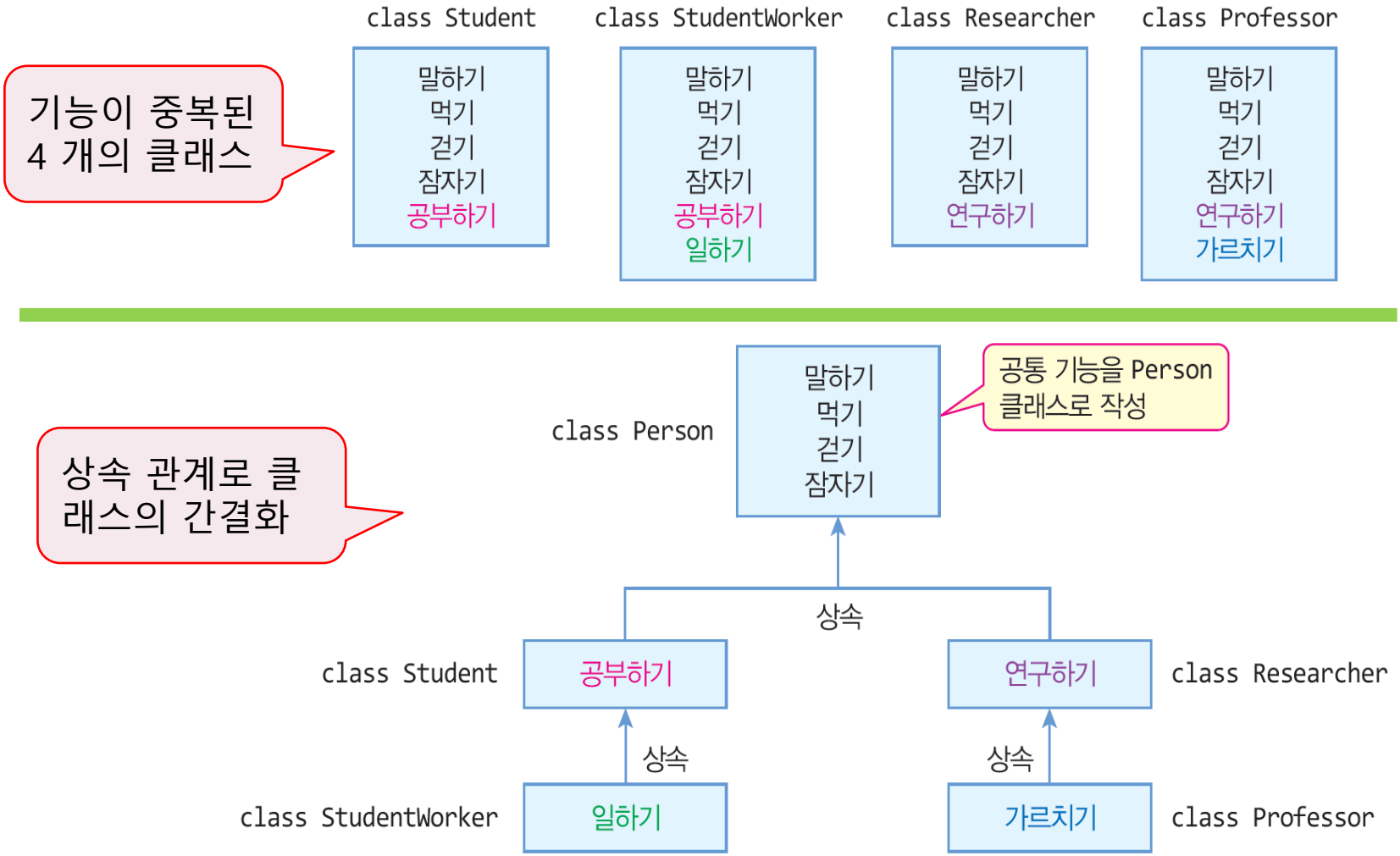
- 프로그램 내에 **두 개 이상의 클래스가 사용되고,**
- 해당 클래스들에 **공통된 내용이 존재하며,**
- 해당 클래스들에 **서로 다른 내용이 존재할 경우**

■ 상속의 목적 및 장점

- **간결한 클래스 작성**
  - 기본 클래스의 기능을 물려받아 파생 클래스를 간결하게 작성
- **클래스 간의 계층적 분류 및 관리의 용이**
  - 상속은 클래스들의 구조적 관계 파악 용이
- **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

클래스간의 상속

■ 상속 관계로 클래스의 간결화 사례



클래스간의 상속

■ 상속 선언 방법

■ 기본 클래스와 파생 클래스

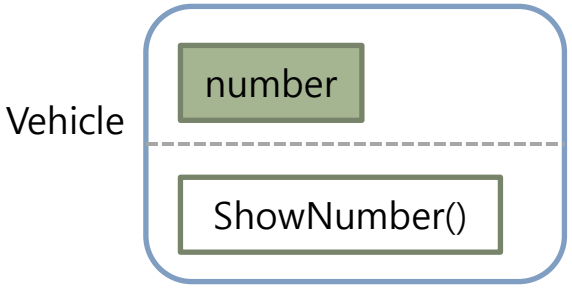
- 파생 클래스의 정의에서 상속관계 표현

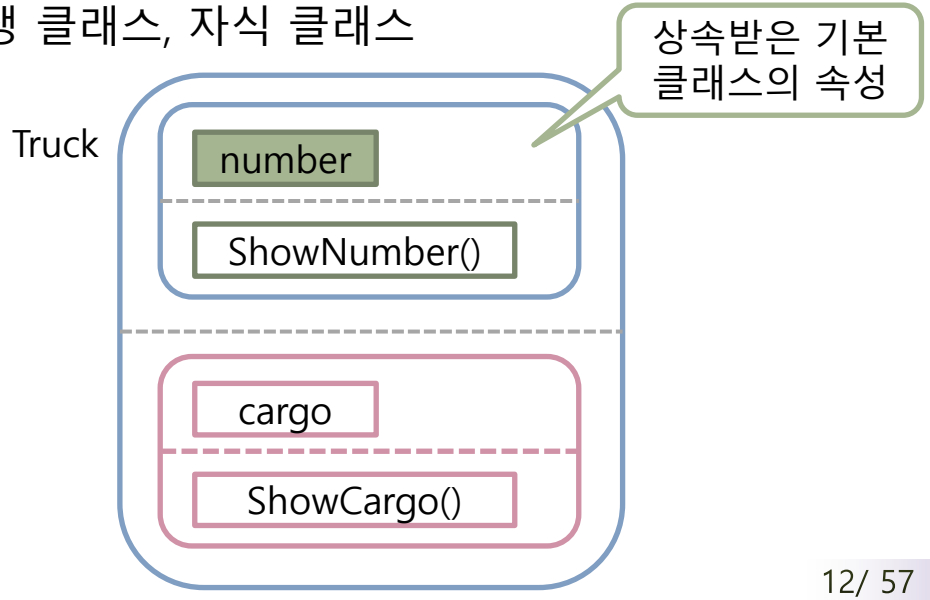
```
class  파생_클래스_이름 : 접근_변경자  기본_클래스_이름
{
  접근_지정자 :
        추가하는_멤버;
        …
};
```

클래스간의 상속

■ 기본 클래스와 파생 클래스 예시

```
class Vehicle                    // 기본 클래스, 부모 클래스
{
private :
        int   number;
public :
        void   ShowNumber();
};

class Truck : public Vehicle     // 파생 클래스, 자식 클래스
{
private :
        int   cargo;
public :
        void   ShowCargo();
};
```





클래스간의 상속

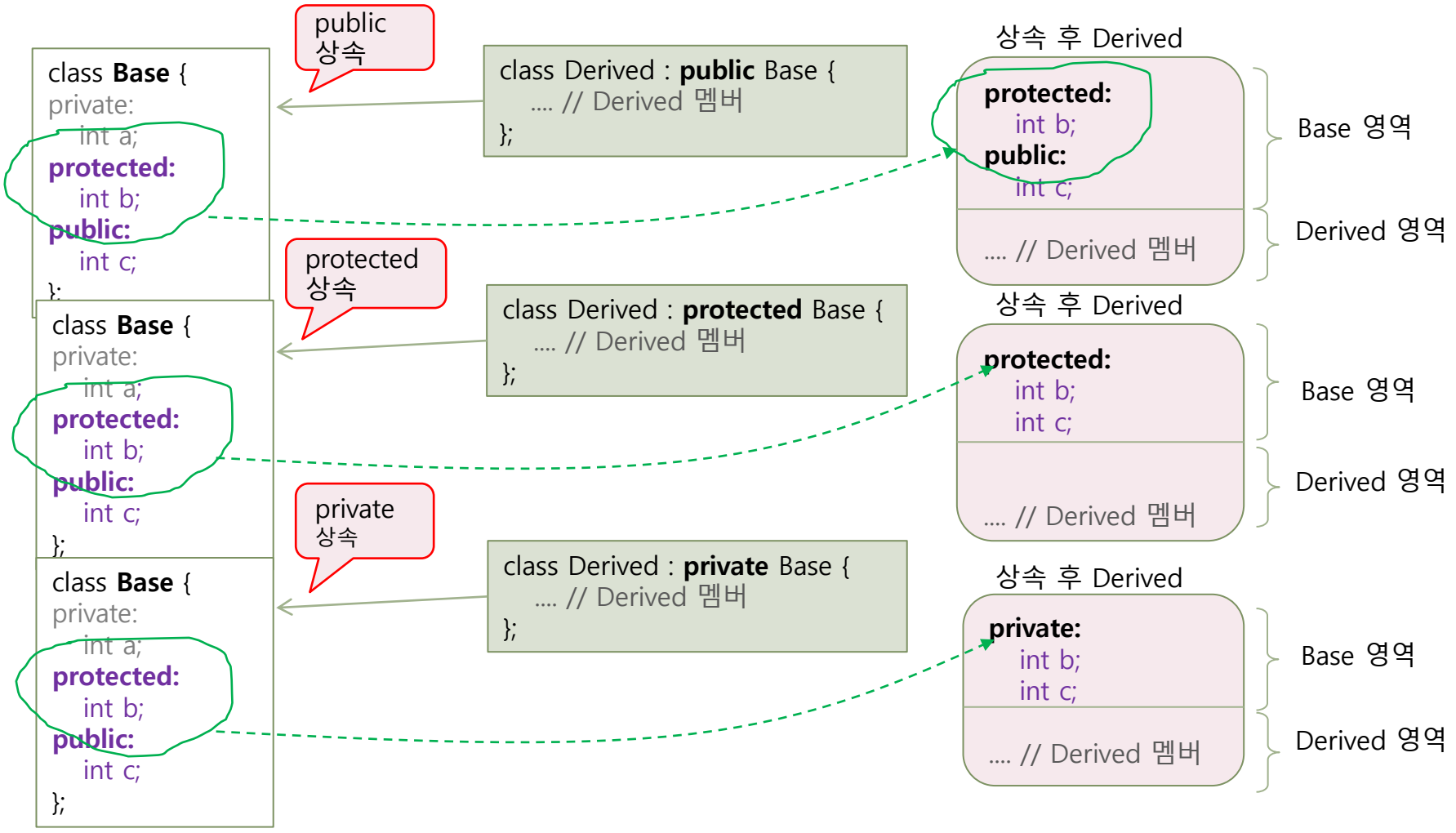
■ 상속의 형태 지정(접근 제한자(변경자))

- 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

접근 제한자 (접근 변경자)	부모 클래스		
	public 멤버	protected 멤버	private 멤버
private 상속	private로 변경	private로 변경	접근 안 됨(X)
protected 상속	protected로 변경	protected	접근 안 됨(X)
public 상속	public	protected	접근 안 됨(X)

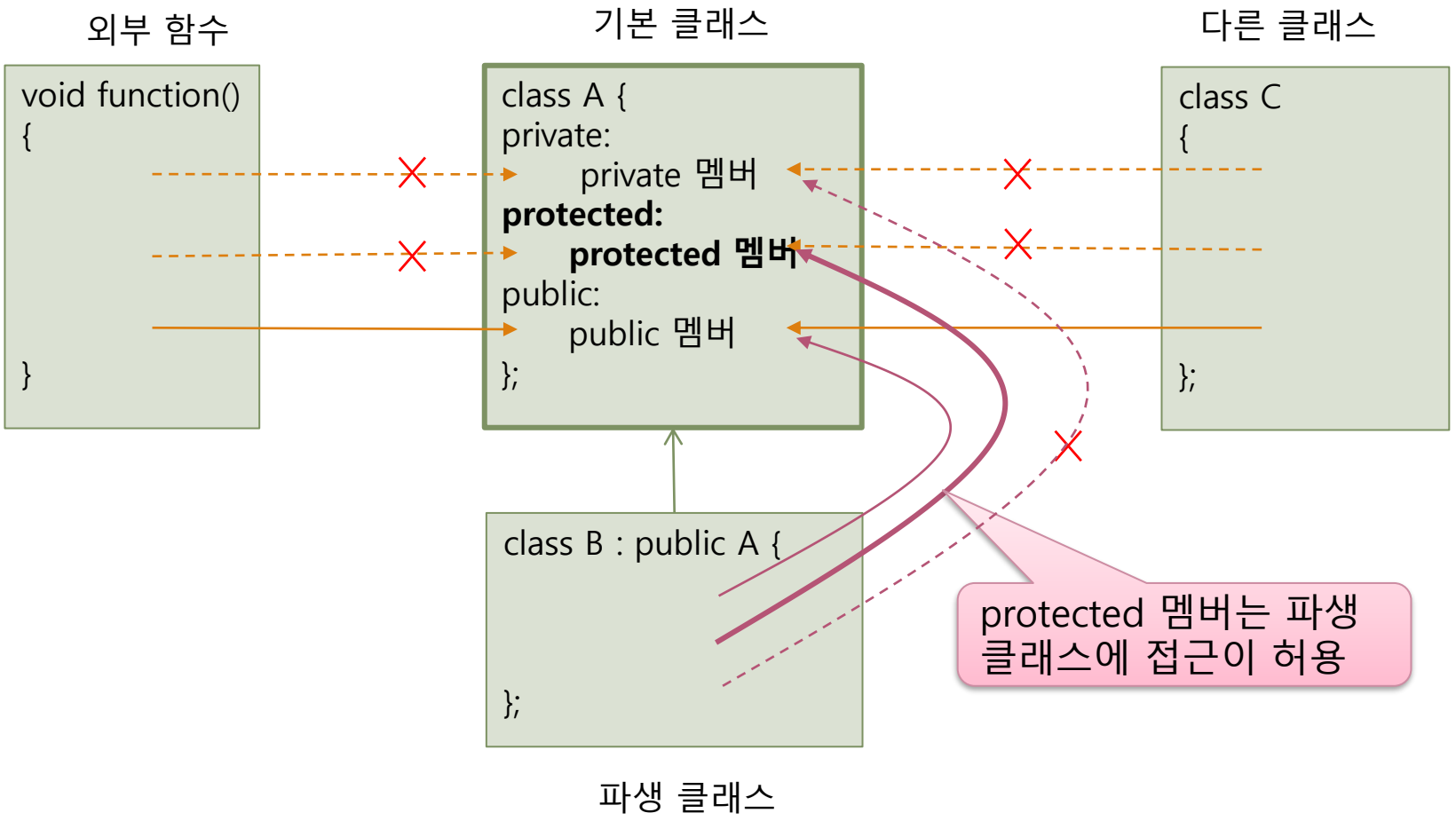
클래스간의 상속

- **상속에서의 접근 제한자에 따른 멤버의 접근 지정 속성 변화**



클래스간의 상속

- **멤버의 접근 지정에 따른 접근성**



클래스간의 상속

■ 멤버의 접근 지정에 따른 접근성 예1

```
class BaseClass{
private:
        int a;
protected:
        int b;
public:
        int c;
};

class DerivedClass : public BaseClass{
public:
        DerivedClass(int i, int j, int k){
                a = i;
                b = j;
                c = k;
        }
        void ShowData(){
                cout << a << "₩t" << b << "₩t"
                << c << endl;
        }
};
```

오류발생

오류발생

```
void main(){
        DerivedClass dc(10, 20, 30);
        dc.ShowData();

        dc.a = 100;
        dc.ShowData();

        dc.b = 200;
        dc.ShowData();

        dc.c = 300;
        dc.ShowData();
}
```

오류발생

오류발생

클래스간의 상속

■ 멤버의 접근 지정에 따른 접근성 예2

```
class BaseClass{
private:
        int a;
protected:
        int b;
public:
        int c;
};

class DerivedClass : protected BaseClass{
public:
        DerivedClass(int i, int j, int k){
                a = i;
                b = j;
                c = k;
        }
        void ShowData(){
                cout << a << "₩t" << b << "₩t" << c << endl;
        }
};
```

오류발생

오류발생

```
void main(){
        DerivedClass dc(10, 20, 30);
        dc.ShowData();

        dc.a = 100;
        dc.ShowData();

        dc.b = 200;
        dc.ShowData();

        dc.c = 300;
        dc.ShowData();
}
```

오류발생

오류발생

오류발생

클래스간의 상속

■ 멤버의 접근 지정에 따른 접근성 예3

```
class BaseClass{
private:
        int a;
protected:
        int b;
public:
        int c;
};

class DerivedClass : private BaseClass{
public:
        DerivedClass(int i, int j, int k){
                a = i;
                b = j;
                c = k;
        }
        void ShowData(){
                cout << a << "₩t" << b << "₩t" << c << endl;
        }
};
```

오류발생

오류발생

```
void main(){
        DerivedClass dc(10, 20, 30);
        dc.ShowData();

        dc.a = 100;
        dc.ShowData();

        dc.b = 200;
        dc.ShowData();

        dc.c = 300;
        dc.ShowData();
}
```

오류발생

오류발생

오류발생

클래스간의 상속

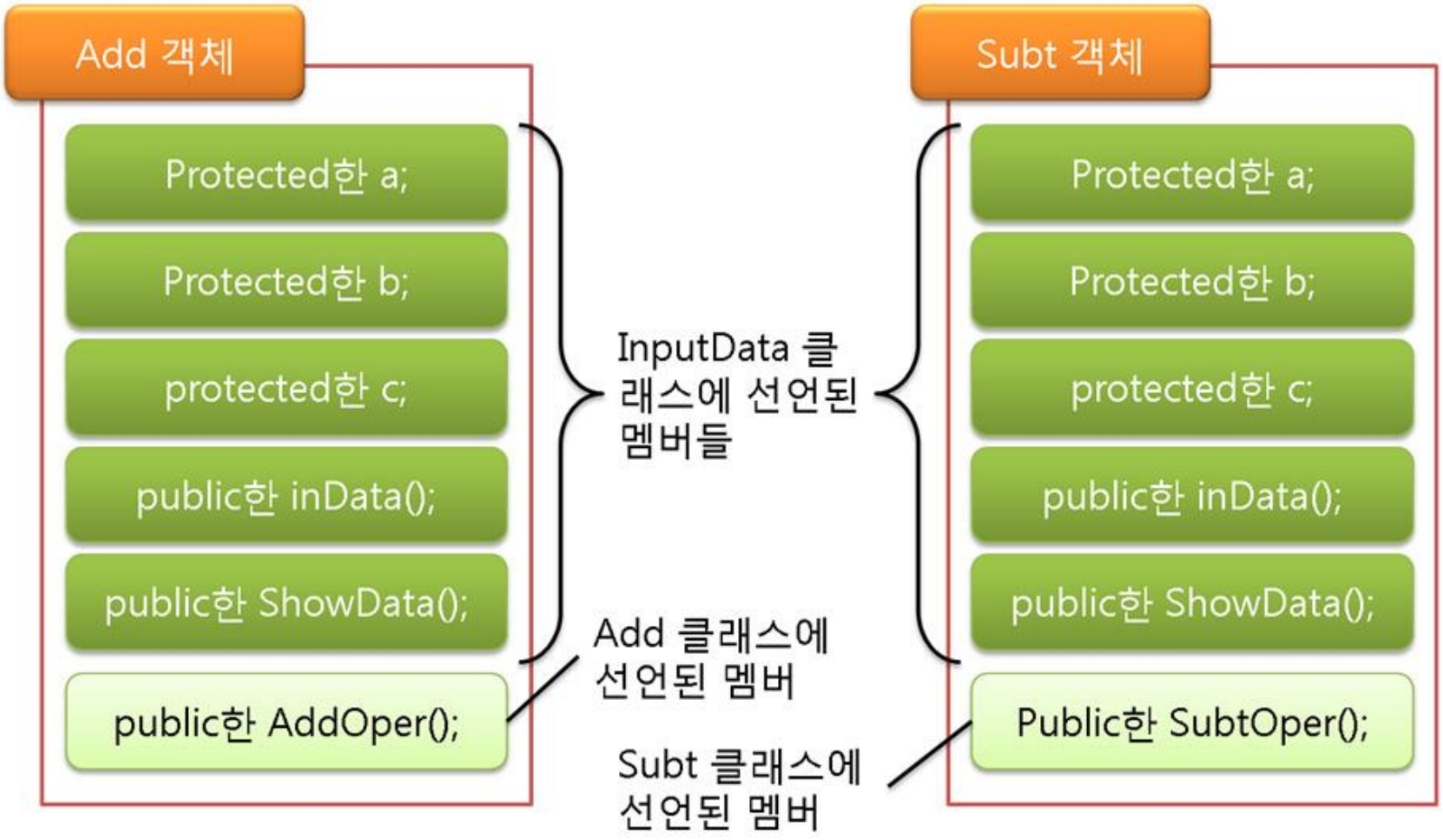
■ 상속 적용 예제(1)

```
#include <iostream>
using std::cout;
using std::endl;
class InputData(){
protected:
        int a;
        int b;
        int c;
public:
        void inData(int, int);
         void ShowData();
};
void InputData::inData(int i, int j){
        a = i;
        b = j;
        c = 0;
}
void InputData::ShowData(){
        cout << a << " , " << b << " : "
        << c << endl;
}
```

```
class Add : public InputData{
public:
        void AddOper();
};
void Add::AddOper(){
        c = a + b;
}
class Subt : public InputData{
public:
        void SubtOper();
};
void Subt::SubtOper(){
        c = a - b;
}
void main(){
        Add in;  Subt sb;
        in.inData(100, 200);
        sb.inData(20, 10);
        in.AddOper();     in.ShowData();
        sb.SubtOper();    sb.ShowData();
}
```

클래스간의 상속

- **상속 적용 예제(2)**



클래스간의 상속

- **상속이 중첩될 때 접근 지정자 사용시 문제 발생 부분은?**

```
#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
   }
};
```

클래스간의 상속

- **질문 1**
  - 파생 클래스의 객체가 생성될 때 파생 클래스의 생성자와 기본 클래스의 생성자가 모두 실행되는가? 아니면 파생 클래스의 생성자만 실행되는가?

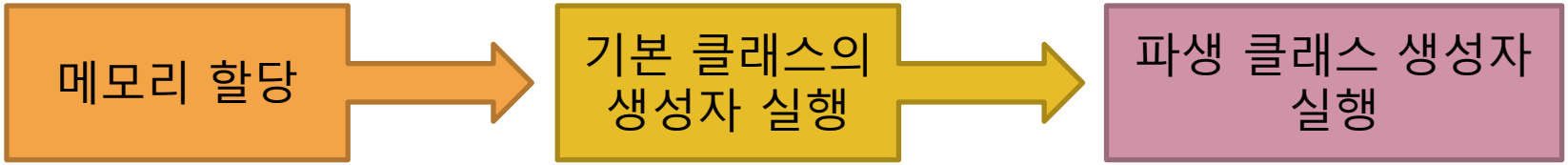
- **질문 2**
  - 파생 클래스의 생성자와 기본 클래스의 생성자 중에서 어떤 생성자가 먼저 실행되는가?

클래스간의 상속

■ 상속에서 생성자와 소멸자 문제

- 상속 관계에 있는 클래스에서 생성자와 소멸자는 멤버함수이지만 상속이 되지 않음
- 파생 클래스에서 생성자와 소멸자를 별도로 만들어야 함

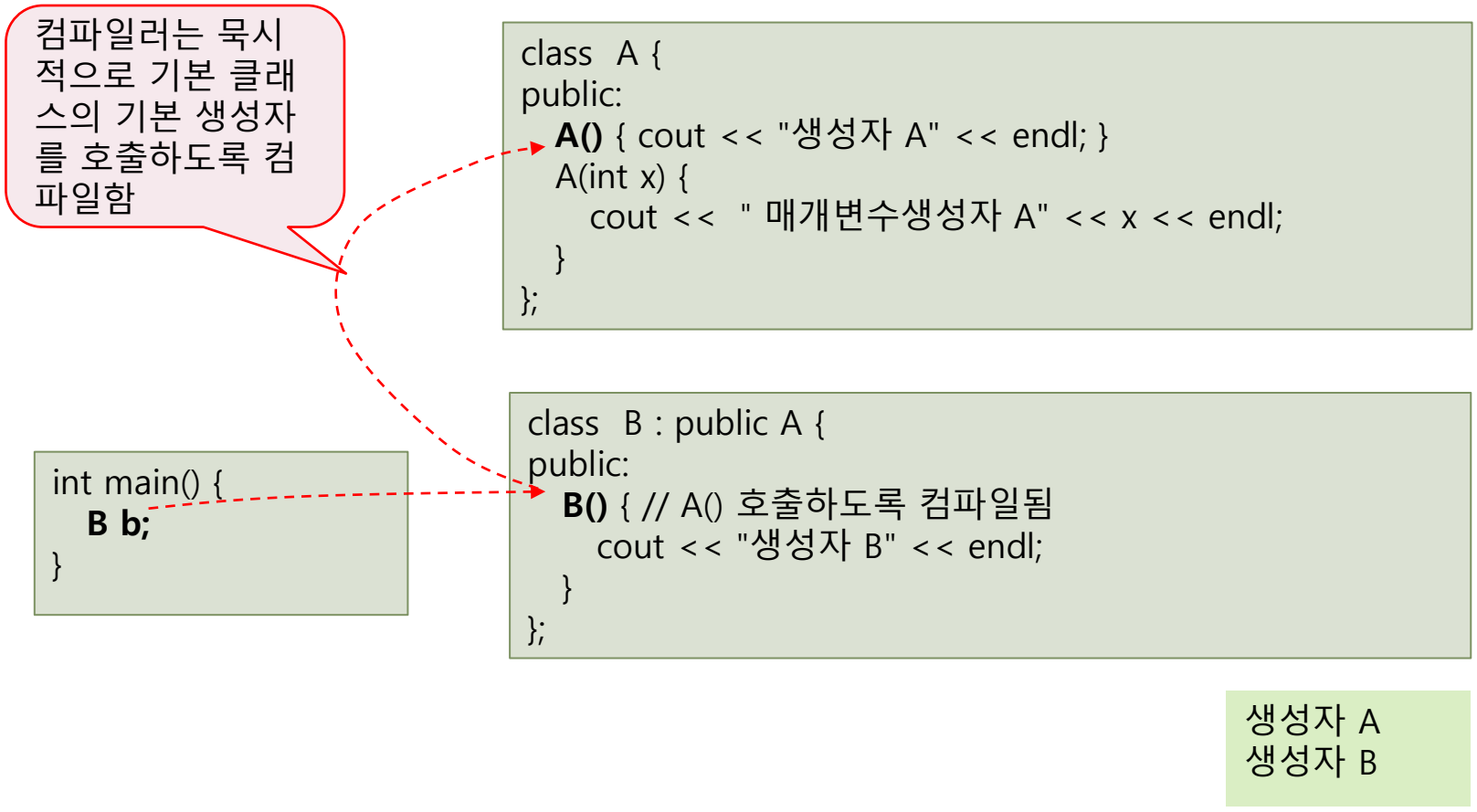
■ 객체 생성과정



- 파생 클래스에서 객체를 생성시
  - 기본 클래스의 생성자가 먼저 호출되고
    - 기본 클래스의 멤버변수를 위한 기억공간을 할당
  - 그 다음에 파생 클래스의 생성자가 호출된다
    - 파생 클래스의 멤버변수를 위한 기억공간을 할당

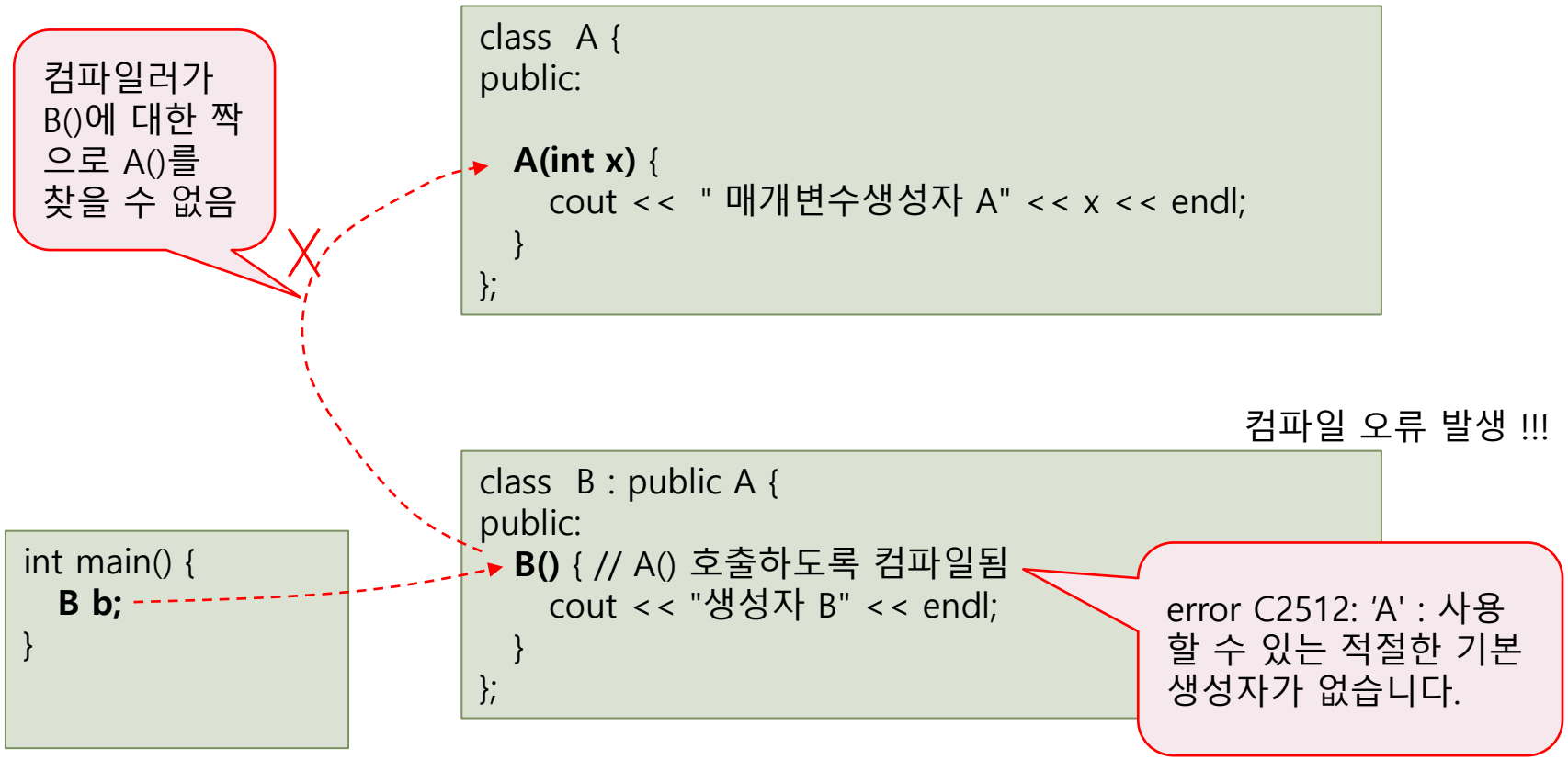
클래스간의 상속

- **컴파일러에 의해 묵시적으로 기본 클래스의 생성자를 선택하는 경우**
  - 파생 클래스의 생성자에서 기본 클래스의 기본 생성자 호출



클래스간의 상속

■ 기본 클래스에 기본 생성자가 없는 경우



클래스간의 상속

- **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
```

클래스간의 상속

- **파생 클래스의 생성자에서 명시적으로 기본 클래스의 생성자 선택**

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;
};
```

매개변수생성자 A8

매개변수생성자 B5

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;
   }
};
```

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

B(5) 호출

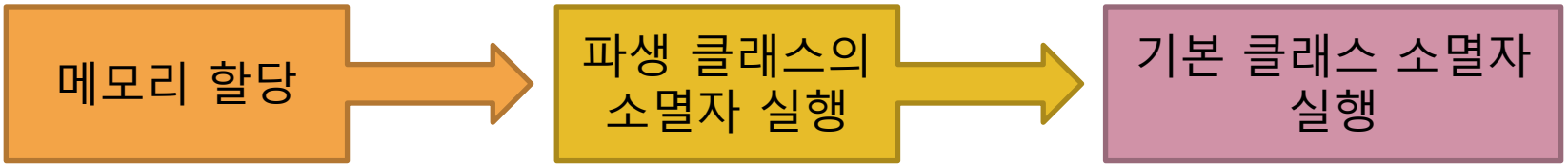
```
class B {
   B() : A() {
      cout << "생성자 B" << endll;
   }
   B(int x) : A() {
      cout << "매개변수생성자 B" << x << endll;
   }
};
```

컴파일러가 묵시적으로 삽입한 코드

컴파일러가 묵시적으로 삽입한 코드

클래스간의 상속

객체 소멸과정



소멸자 호출 순서

- 파생 클래스의 소멸자가 먼저 호출되고
- 그 다음에 기본 클래스의 소멸자가 호출
- 생성자 호출 순서와 반대임

클래스간의 상속

ex9_1.cpp (1) (public으로 상속받는 클래스)

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

class Vehicle                                   // 기본 클래스
{
private:
        int number;
public:
        Vehicle(int n):number(n) {              // 기본 클래스의 생성자
                cout << "Vehicle 생성자" << endl;
        };
        ~Vehicle() {                            // 기본 클래스의 소멸자
                cout << "Vehicle 소멸자, " << number << endl;
        };
        void ShowNumber() const ;
};
void Vehicle::ShowNumber() const
{
        cout << "번호: " << number << endl;
}
```

클래스간의 상속

ex9_1.cpp (2) (public으로 상속받는 클래스)

```
class Truck:public Vehicle                    // 파생 클래스
{
private:
        int cargo;
public:
        Truck(int n, int c):Vehicle(n)        // 파생 클래스의 생성자
         {
                cargo = c;
                cout << "Truck 생성자" << endl;
        }

        ~Truck()                              // 파생 클래스의 소멸자
        {
                cout << "Truck 소멸자, " << cargo << endl;
        }
        void ShowCargo() const ;
};
void Truck::ShowCargo() const
{
        cout << "화물: " << cargo << endl;
}
```

클래스간의 상속

ex9_1.cpp (3) (public으로 상속받는 클래스)

```
void main()
{
	Vehicle v(1234);
	v.ShowNumber();
	cout << endl;

	Truck t(7890, 5);		// Truck은 public 상속
	t.ShowNumber();
	t.ShowCargo();
	cout << endl;
}
```

```
Vehicle 생성자
번호: 1234

Vehicle 생성자
Truck 생성자
번호: 7890
화물: 5

Truck 소멸자, 5
Vehicle 소멸자, 7890
Vehicle 소멸자, 1234
계속하려면 아무 키나 누르십시오 . . .
```

클래스간의 상속

ex9_2.cpp (1) (protected로 상속받는 클래스)

```
#include <iostream>
using namespace std;
class Vehicle
{
private:
        int number;
public:
        Vehicle(int n):number(n) { cout << "Vehicle 생성자" << endl;};
        void Show() const ;
};
void Vehicle::Show() const
{
        cout << "번호: " << number << endl;
}
class Truck:protected Vehicle
{
private:
        int cargo;
public:
        Truck(int n, int c);
        void displayTruck();
};
```

클래스간의 상속

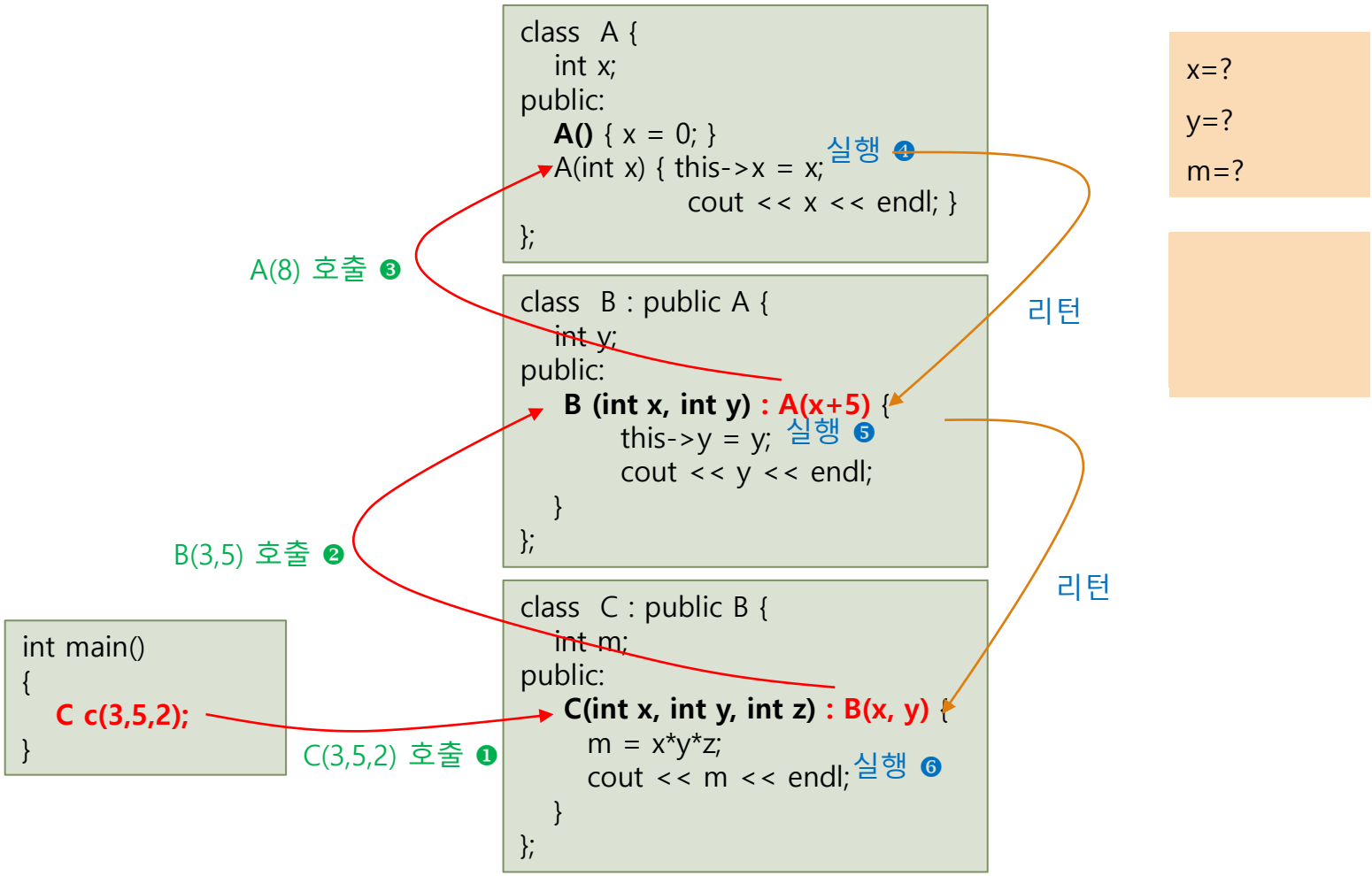
ex9_2.cpp (2) (protected로 상속받는 클래스)

```
Truck::Truck(int n, int c):Vehicle(n)
{
        cout << "Truck 생성자" << endl;
        cargo = c;
}
void Truck::displayTruck()
{
        Show();
        cout << "화물 적재량: " << cargo << endl;
}
void main()
{
        Vehicle v(1234);
        v.Show();
        cout << endl;
        Truck t(7890, 5);          // Truck은 protected 상속
//      t.Show();                  // t.Show(): 클래스 외부에서 접근 불가
        t.displayTruck();
        cout << endl;
}
```

```
Vehicle 생성자
번호: 1234

Vehicle 생성자
Truck 생성자
번호: 7890
화물 적재량: 5

계속하려면 아무 키나 누르십시오 . . .
```

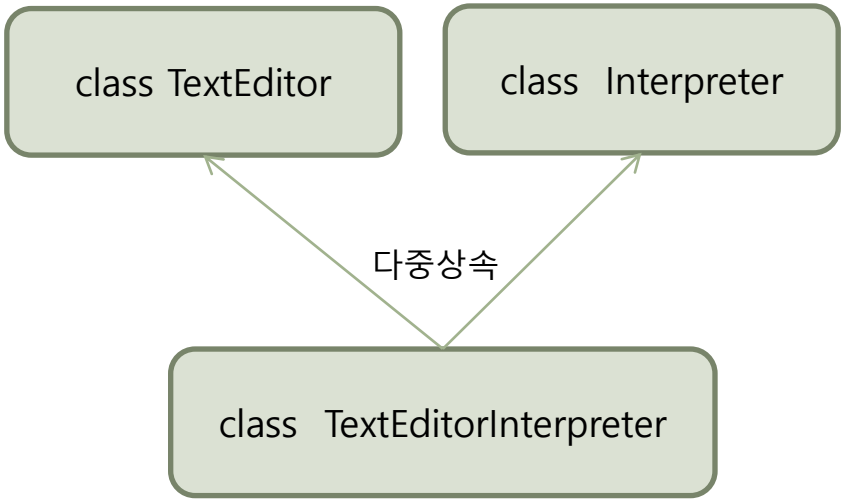
클래스간의 상속

- 상속 관계의 생성자 매개 변수 처리시 결과는?



클래스간의 상속

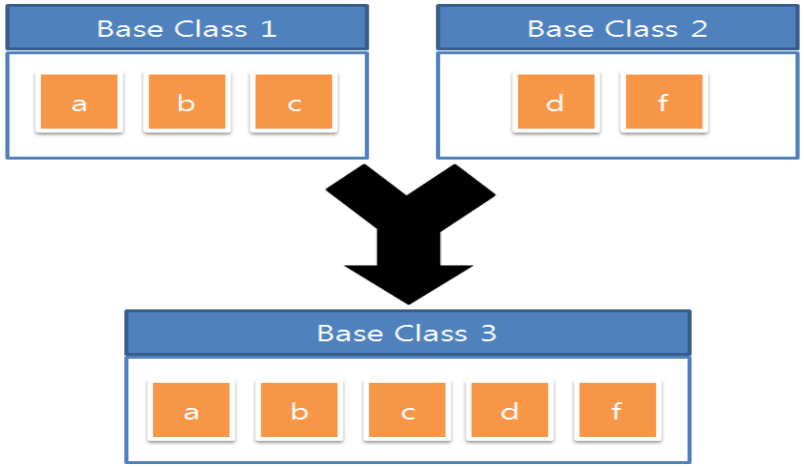
- **다중 상속(multiple inheritance)**
  - 자식 클래스가 두 개 이상의 부모 클래스로부터 멤버를 상속받는 것
  - 다중 상속은 유익한 이점도 있지만 잘못 사용할 경우가 많으므로 사용에 주의하여야 함



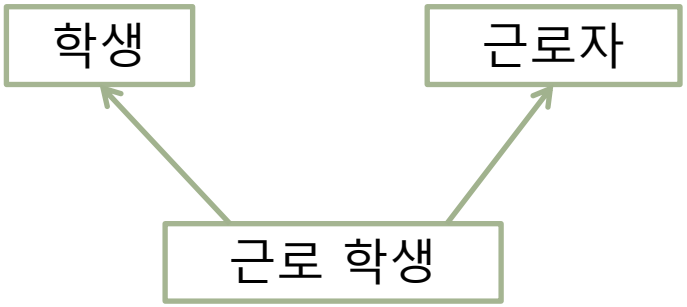
클래스간의 상속

■ 다중 상속(multiple inheritance)

- 다중 상속은 여러 개의 기본 클래스를 동시에 상속한 파생 클래스를 만드는 것



[예]
```
class Student { ... };
class Worker {... };
class Wstudent :  public Student, public Worker { ... };
```



클래스간의 상속

다중 상속의 정의

- 파생 클래스를 정의할 때, 상속할 기본 클래스의 이름을 모두 나열
- 클래스의 이름을 **“,”(콤마)로 구분하여 정의**

다중 상속 형식

```
class Derived : public Base1, Base2
{
        ...
}
```

다중 상속에서의 멤버 활용

- 같은 이름의 멤버 중 어떤 기본 클래스에 선언되었던 멤버를 호출하는 것인지를 **스코프 연산자**(::)를 사용하여 구분해서 활용

```
da.Triangle::ShowData();
```

클래스간의 상속

- **ex9_3.cpp (1) (다중 상속의 예)**

```
#include <iostream>
#include <cstring>
using namespace std;
class Student
{
protected:
	int id;
	char *name;
	int score;
public:
	Student(int num, char *na, int sco);
	void showStudent() {
		cout << "id: " << id << ", name: " << name << ", score: " << score;
	}
};
Student::Student(int num, char *na, int sco)
{
	id = num;
	score = sco;
	name = new char[strlen(na)+1];
	strcpy_s(name, strlen(na)+1, na);
}
```

클래스간의 상속

- **ex9_3.cpp (2) (다중 상속의 예)**

```
class Worker
{
protected:
        int salary;

public:
        Worker(int sal) : salary(sal) { };
        void showSalary() {
                cout << "salary: " << salary;
        }
};
class Wstudent :public Student, public Worker
{
public:
        Wstudent(int num, char* na, int sco, int sal);
};
Wstudent::Wstudent(int num, char* na, int sco, int sal):Student(num, na, sco),Worker(sal)
{

};
```

클래스간의 상속

ex9_3.cpp (3) (다중 상속의 예)

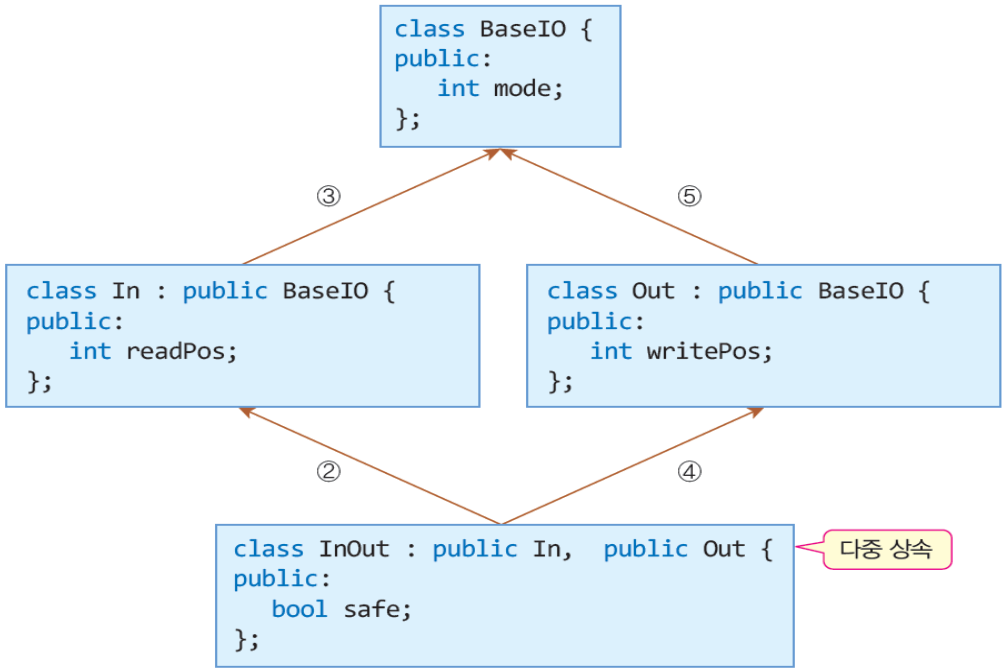
```
void main()
{
	Wstudent wstu1(1234, "Hong Gildong", 87, 1850000);

	cout << "wstu1 ==> " ;
	wstu1.showStudent() ;
	cout << ", " ;

	wstu1.showSalary();
	cout << endl;
}
```

```
wstu1 ==> id: 1234, name: Hong Gildong, score: 87, salary: 1850000
계속하려면 아무 키나 누르십시오 . . .
```

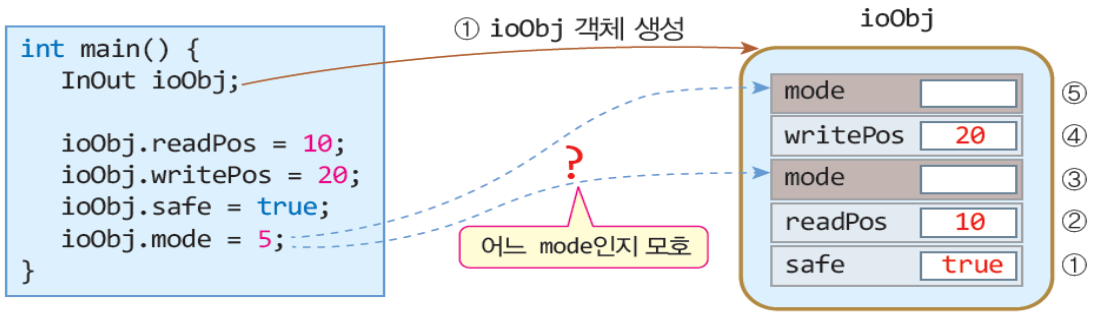
클래스간의 상속

■ 다중 상속의 문제점

- Base의 멤버가 이중으로 객체에 삽입되는 문제점
- 동일한 x를 접근하는 프로그램이 서로 다른 x에 접근하는 결과를 낳게 되어 잘못된 실행 오류가 발생



(a) 클래스 상속 관계



(b) ioObj 객체 생성 과정 및 객체 내부

멤버 함수의 재정의

■ 함수의 재정의(function overriding)

- 기본 클래스의 멤버함수와 같은 이름의 함수를 인수의 자료형이나 개수에 관계 없이 **파생 클래스에서도 같은 이름의 멤버함수로 다시 정의 가능**
- 함수의 재정의를 사용하여 자식 클래스는 부모 클래스로부터 **상속받은 멤버함수를 자기 자신의 필요에 맞게 변경하여 사용할 수 있음**
- 동일한 이름의 멤버함수가 기본 클래스와 파생 클래스 모두에 정의되어 있는 경우,
  - 파생 클래스의 객체가 함수를 호출하면 파생 클래스에 있는 함수가 우선권이 있어 파생 클래스의 함수가 호출
  - 파생 클래스의 객체가 부모 클래스의 멤버함수를 호출하려면 영역 지정 연산자 ::를 사용하여 "부모_클래스::"을 함수 앞에 붙여 사용

멤버 함수의 재정의

- **함수의 다중 정의(function overloading)**
  - 같은 이름의 함수를 사용하여, 하나의 이름으로 여러 가지 기능 제공
  - 함수의 인수 자료형이나 인수가 다른 경우로 함수가 각각 별도로 존재하는 경우 사용

- **함수의 재정의(function overriding)**
  - 함수를 다른 기능으로 다시 정의하는 경우 사용
  - 함수의 인수가 같을 수도 있음

멤버 함수의 재정의

ex9_4.cpp (1) (멤버 함수의 재정의)

```
#include <iostream>
using namespace std;
class Vehicle
{
protected:
        int number;
public:
        Vehicle(int n) : number(n) {    };
        void Show() const ;
};
void Vehicle::Show() const  {
        cout << "Vehicle::Show() ==> " ;
        cout << "번호: " << number << endl;
}
class Truck : public Vehicle
{
private:
        int cargo;
public:
        Truck(int n, int c):Vehicle(n) { cargo = c; } ;
        void Show() const ;
};
```

멤버 함수의 재정의

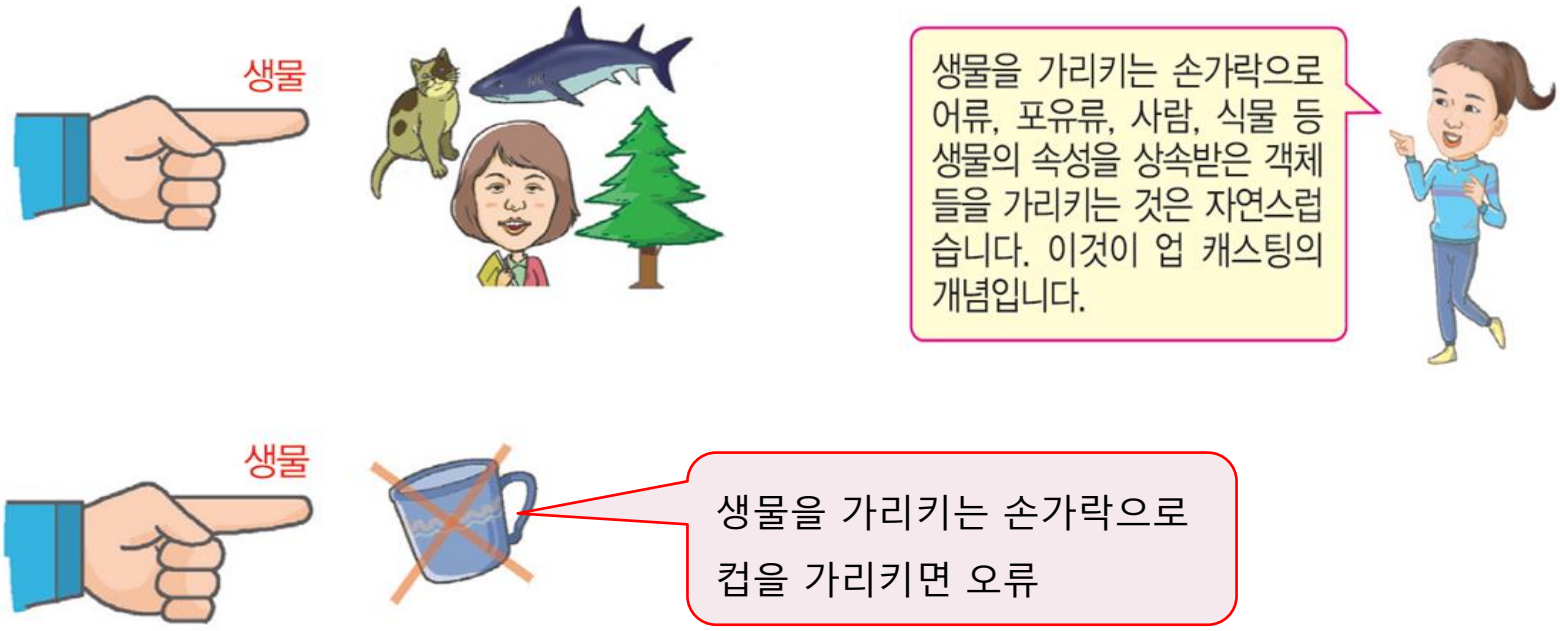
■ ex9_4.cpp (2) (멤버 함수의 재정의)

```
void Truck::Show() const
{
        cout << "Truck::Show() ==> " ;
        cout << "번호: " << number << ", 화물 적재량: " << cargo << endl;
}
void main()
{
        Vehicle v(1234);
        cout << "v ==> " ;
        v.Show();

        Truck t(3456, 40);
        cout << "t ==> " ;
        t.Show();

        cout << "t ==> " ;
        t.Vehicle::Show();
        cout << endl;
}
```

```
v ==> Vehicle::Show() ==> 번호: 1234
t ==> Truck::Show() ==> 번호: 3456, 화물 적재량: 40
t ==> Vehicle::Show() ==> 번호: 3456

계속하려면 아무 키나 누르십시오 . . .
```

부모-자식 클래스간의 형 변환

- **업 캐스팅(up-casting)**
  - 파생 클래스의 객체를 기본 클래스의 포인터로 가리키는 것
  - 파생 클래스의 객체를 기본 클래스의 객체처럼 다룰 수 있는 것
- **파생 클래스 포인터가 기본 클래스 포인터에 치환되는 것**
  - 예) 사람을 동물로 봄



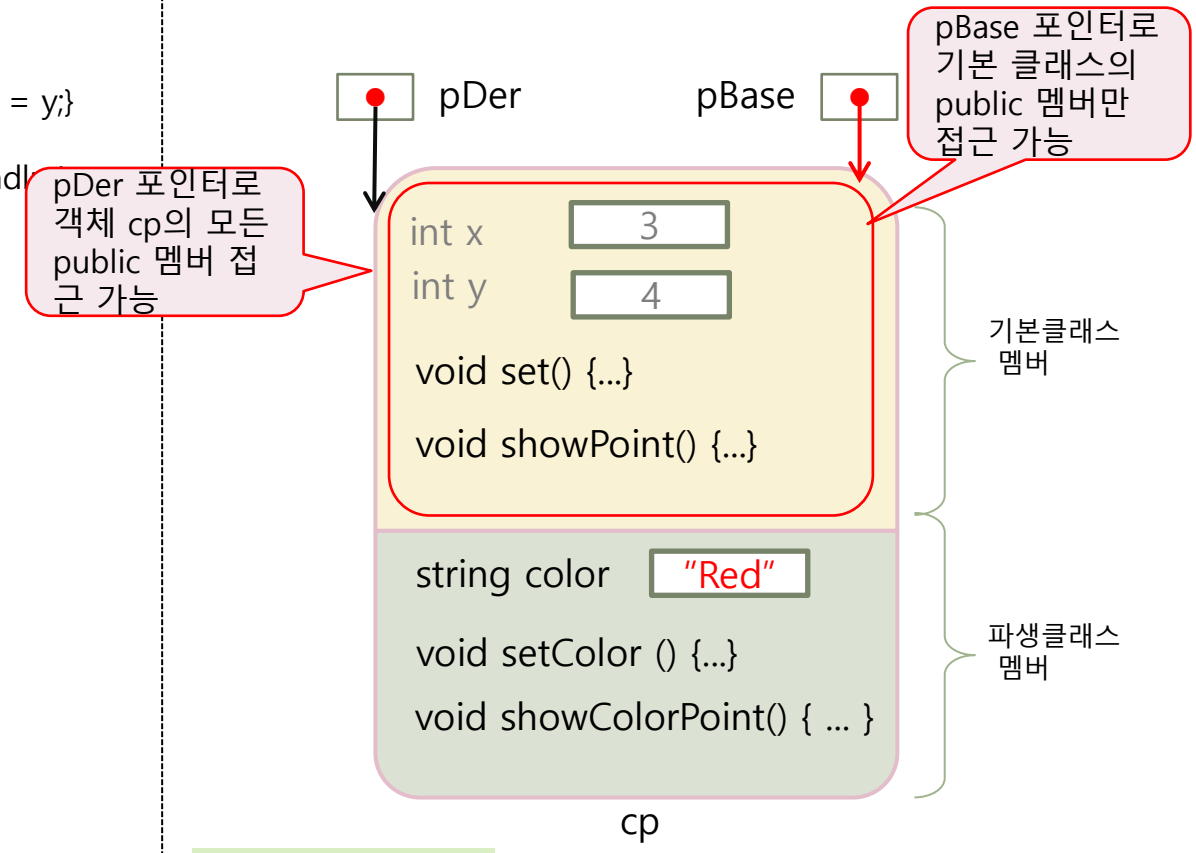
부모-자식 클래스간의 형 변환

- **업 캐스팅(up casting)**
  - 자식 클래스의 객체를 부모 클래스의 객체에 대입하는 것
  - 자식 클래스의 포인터나 참조를 부모 클래스의 포인터나 참조로 변환하는 것
  - public 상속에서는 명시적이 아니라도 업 캐스팅은 항상 허용
    - 자식-부모의 관계는 is-a 관계에 의한 규칙 수용
    - 자식 클래스의 객체에만 있는 멤버 변수들은 부모 클래스의 객체의 멤버 변수들에게 아무 영향을 줄 수 없기 때문

부모-자식 클래스간의 형 변환

■ 업 캐스팅(up-casting) 예시

```
class Point {
    int x, y;
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;
public:
    void setcolor(string color){
            this->color=color;}
    void showColorPoint() {
        cout<<color <<" : ";
        showPoint(); }
};
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)

부모-자식 클래스간의 형 변환

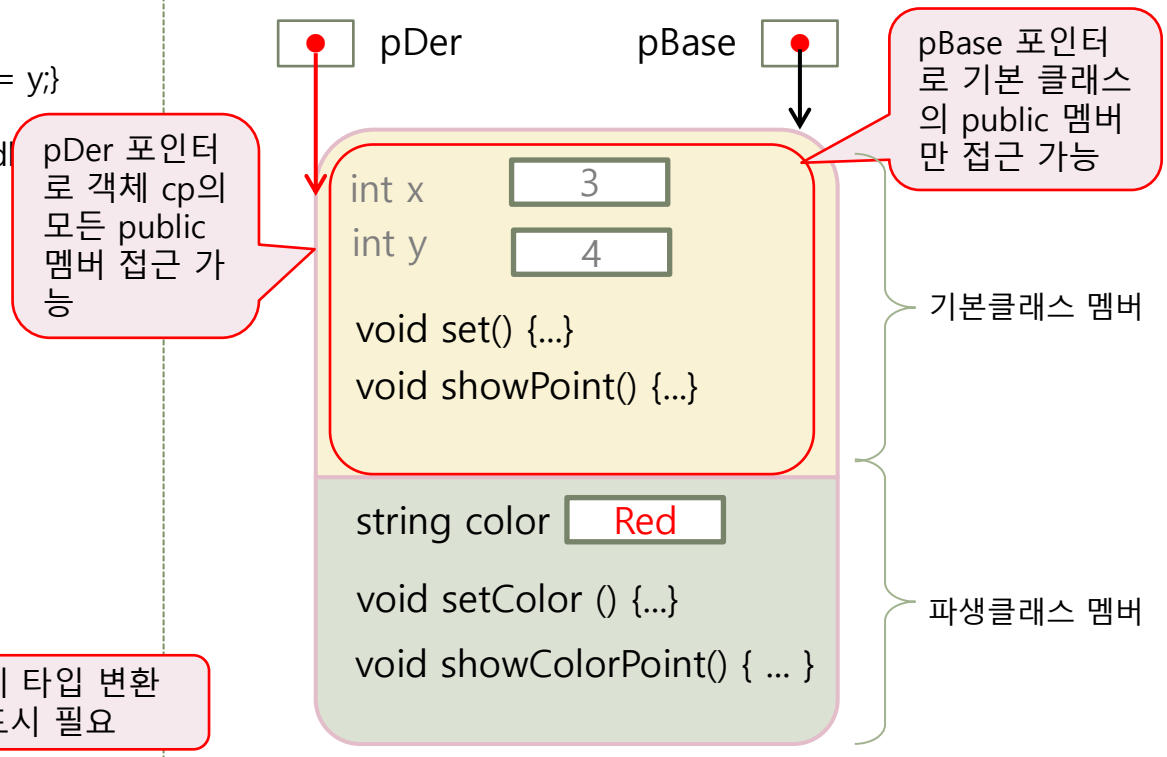
- **다운 캐스팅(down casting)**
  - 부모 클래스 객체를 자식 클래스 객체에 대입하는 것
  - 부모 클래스의 포인터에서 자식 클래스의 포인터로 형 변환하는 것
  - 다운 캐스팅은 명시적인 형 변환 없이는 허용되지 않음
    - 일반적으로 is-a 관계는 역으로는 성립하지 않기 때문
    - 자식 클래스의 객체에만 있는 멤버 변수들에게 값을 줄 수가 없기 때문

부모-자식 클래스간의 형 변환

■ 다운 캐스팅(down-casting) 예시

```
class Point {
    int x, y;
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;
public:
    void setcolor(string color){
            this->color=color;}
    void showColorPoint() {
        cout<<color <<" : ";
        showPoint(); }
};
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
}
```

강제 타입 변환 반드시 필요

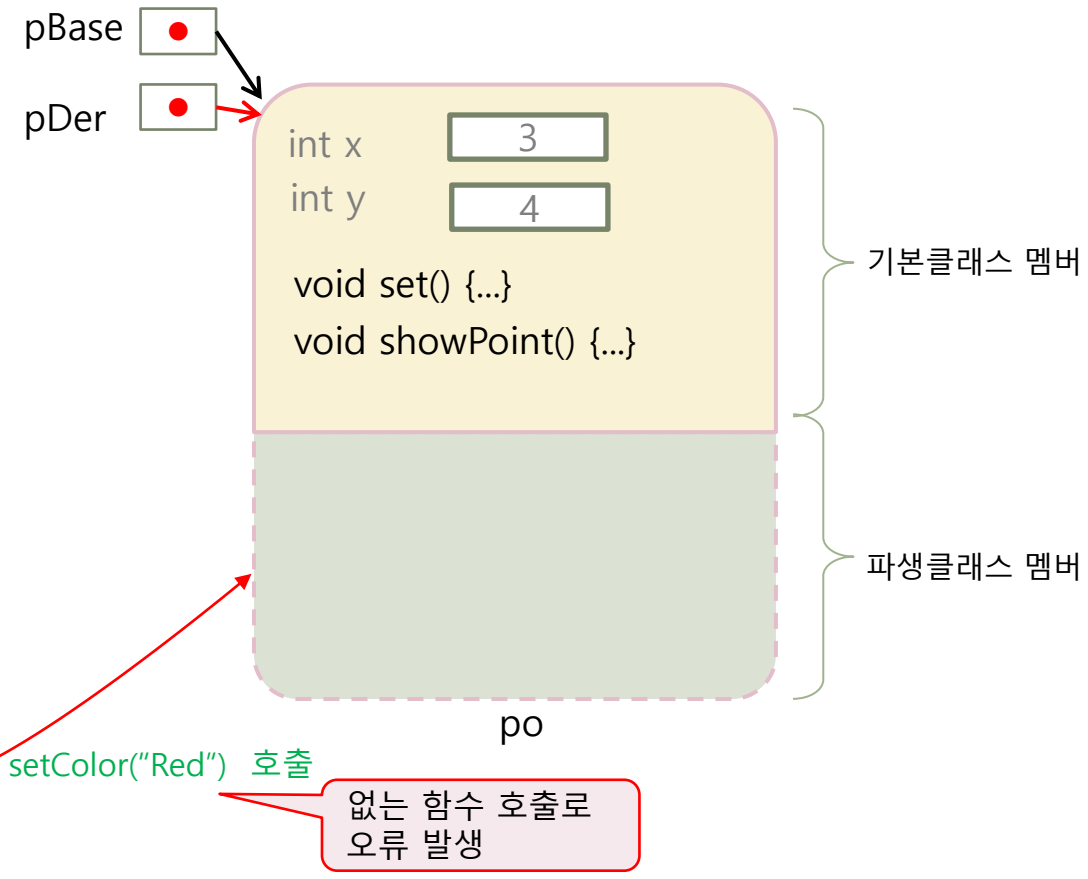


(3,4)
Red(3,4)

부모-자식 클래스간의 형 변환

■ 다운 캐스팅(down-casting)의 주의점

```
class Point {
    int x, y;
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;
public:
    void setcolor(string color){
            this->color=color;}
    void showColorPoint() {
        cout<<color <<" : ";
        showPoint(); }
};
int main() {
    ColorPoint *pDer;
    Point *pBase, po;
    pBase = &po;
    pDer = (ColorPoint *)pBase;  //다운 캐스팅

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



부모-자식 클래스간의 형 변환

■ 부모-자식 클래스간의 형 변환 예

```
class Vehicle                       // 기본 클래스, 부모 클래스
{
private :
        int number;
public :
        void ShowNumber();
};

class Truck : public Vehicle        // 파생 클래스, 자식 클래스
{
private :
        int cargo;
public :
        void ShowCargo();
};
```

부모-자식 클래스간의 형 변환

- **포인터간의 형, 참조간의 형 변환 예**

```
Vehicle  v1;
Truck    t1;

t1 = v1;                // 오류
v1 = t1;                // 가능

Truck *pt = &v1;        // 다운 캐스팅 - 오류
Truck &rt = v1;         // 다운 캐스팅 – 오류

Vehicle *pv = &t1;      // 업 캐스팅 - 허용
Vehicle &rv = t1;       // 업 캐스팅 - 허용
```

부모-자식 클래스간의 형 변환

- **ex9_5.cpp (1) (부모 객체와 자식 객체간의 대입)**

```
#include <iostream>
using namespace std;
class Vehicle
{
private:
	int number;
public:
	Vehicle(int n):number(n) {	}
	void showNumber() const
	{
		cout << "번호: " << number << endl;
	}
};
class Truck : public Vehicle
{
private:
	int cargo;
public:
	Truck(int n, int c):Vehicle(n) {		cargo = c;		}
	void showCargo() const {	cout << "화물: " << cargo << endl;	}
};
```

부모-자식 클래스간의 형 변환

ex9_5.cpp (2) (부모 객체와 자식 객체간의 대입)

```
void main()
{
        Vehicle v1(1234);
        Truck t1(5678, 5);

        cout << "v1 --> " << endl;
        v1.showNumber();

        cout << "------------------" << endl;
        cout << "t1 --> " << endl;
        t1.showNumber();
        t1.showCargo();

        cout << "------------------" << endl;
//      t1 = v1;                    // 오류
        cout << "v1 = t1 --> " << endl;

        v1 = t1;                    // 허용
        v1.showNumber();
}
```

```
v1 -->
번호: 1234
------------------
t1 -->
번호: 5678
화물: 5
------------------
v1 = t1 -->
번호: 5678
계속하려면 아무 키나 누르십시오 . . .
```

부모-자식 클래스간의 형 변환

- **ex9_6.cpp (1) (부모 객체와 자식 포인터간, 참조간의 대입)**

```
#include <iostream>
using namespace std;
class Vehicle
{
private:
	int number;
public:
	Vehicle(int n):number(n) {	}
	void showNumber() const {	cout << "번호: " << number << endl;	}
};

class Truck : public Vehicle
{
private:
	int cargo;
public:
	Truck(int n, int c):Vehicle(n) {	cargo = c;	}
	void showCargo() const {	cout << "화물: " << cargo << endl;	}
};
```

부모-자식 클래스간의 형 변환

ex9_6.cpp (2) (부모 객체와 자식 포인터간, 참조간의 대입)

```
void main()
{
        Vehicle v1(1234);
        cout << "v1 --> " << endl;
        v1.showNumber();
        cout << "------------------" << endl;
//      Truck *pt = &v1;                // 다운 캐스팅
//      Truck &rt = v1;                 // 다운 캐스팅

        Truck t1(5678, 5);
        cout << "t1 --> " << endl;
        t1.showNumber();
        t1.showCargo();
        cout << "------------------" << endl;
        Vehicle *pv = &t1;              // 업 캐스팅
        cout << "Vehicle *pv = &t1 --> " << endl;
        pv->showNumber();               // 포인터 형을 기준으로 호출될 함수가 결정된다.
        cout << "------------------" << endl;

        Vehicle &rv = t1;               // 업 캐스팅
        cout << "Vehicle &rv = t1 --> " << endl;
        rv.showNumber();                // 참조를 기준으로 호출될 함수가 결정된다.
}
```

```
v1 -->
번호: 1234
------------------
t1 -->
번호: 5678
화물: 5
------------------
Vehicle *pv = &t1 -->
번호: 5678
------------------
Vehicle &rv = t1 -->
번호: 5678
계속하려면 아무 키나 누르십시오 . . .
```

Thank You