8.1 상속의 개념

상속 객체 지향 언어의 본질적인 특성

소프트웨어의 재사용을 가능하게 함, 동적 바인딩

유전적 상속과 객체지향상속 유전적 상속과 객체지향상속 유사

자바 상속해주는 클래스 super class

상속받는 클래스 sub class

C++ 상속해주는 클래스 parent class

상속받는 클래스 child class

상속의 목적과 장점 간결한 클래스 작성 동일한 코드가 여러 클래스에 중복되면 유지보수가 힘듬

공통기능을 가진 클래스를 작성하고, 다른 클래스에 추가하면서 상속 🡪 클래스들 사이의 중복을 제거해 클래스를 간결하게 구현

클래스 간의 계층적 분류 및 관리의 용이함 클래스를 계층 관계로 표현 🡪 구조적인 관계 파악이 쉬움

클래스 재사용과 확장을 통한 소프트웨어의 생산성 향상 소프트웨어의 생산성 향상

상속주의 기능이 필요하다고 아무 연관없는 클래스를 상속받아선 안됨

check time 1. 3

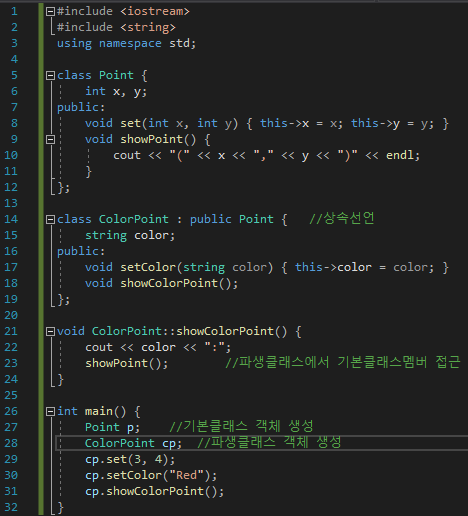
2. 3

8.2 클래스 상속과 객체

상속선언 형식 class 파생클래스명 : 접근지정자 부모클래스 { };

ex) class Student : public Person { };

파생클래스 객체와 멤버호출 예제8-1



상속선언 ex) class ColorPoint : public Point { //상속선언

string color;

public:

void setColor(string color) { this->color = color; }

void showColorPoint();

};

파생클래스 객체 생성 ex) ColorPoint cp; //파생클래스 객체 생성, cp는 7개의 멤버를 가짐

파생클래스 객체는 기본클래스의 멤버를 포함함

파생클래스에서 기본 클래스 멤버 접근 ex) void ColorPoint::showColorPoint() {

cout << color << ":";

showPoint(); //파생클래스에서 기본클래스멤버 접근

}

파생클래스는 상속을 통해 기본클래의 멤버를 자신의 멤버로 받아들임

파생클래스의 멤버들은 기본클래스의 private멤버외의 모든 멤버 접근 가능

기본클래스의 private멤버의 상속과 접근 기본클래스의 private멤버는 기본클래스의 멤버함수로만 접근가능 ∙ 파생클래스의 멤버로는 접근불가

클래스 외부에서파생 클래스의 멤버 호출 파생클래스의 public멤버와 기본클래스의 public멤버에 접근가능

8.3 상속과 객체 포인터

상속을 통해 기본클래스 확장 🡪 파생클래스 객체를 파생클래스 포인터, 기본클래스 포인터로 가리킬 수 있음

업캐스팅 파생클래스의 객체를 기본클래스의 포인터로 가르키는 것

파생클래스의 객체를 기본클래스의 객체 처럼 다룸

ex) 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 { //상속선언

string color;

public:

void setColor(string color) { this->color = color; }

void showColorPoint();

};

void ColorPoint::showColorPoint() {

cout << color << ":";

showPoint(); //파생클래스에서 기본클래스멤버 접근

}

int main() {

ColorPoint cp;

ColorPoint\* pDer = &cp;

Point\* pBase = pDer; //업캐스팅

pDer->set(3, 4);

pBase->showPoint();

pDer->setColor("red");

pDer->showColorPoint();

}

업캐스팅 업캐스팅한 기본클래스의 포인터로는 기본클래스의 멤버만 접근가능

명시적 타입변환 필요 X 🡨 cp객체는 ColorPoint타입이지만 Point타입이기도 하기때문

ex) Point\* pBase = pDer; //업캐스팅

기본클래스 Point타입 포인터 pBase로 ColorPoint객체를 가리킴 but ColorPoint객체 내의 Point클래스멤버에만 접근가능

다운캐스팅 ex) 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 { //상속선언

string color;

public:

void setColor(string color) { this->color = color; }

void showColorPoint();

};

void ColorPoint::showColorPoint() {

cout << color << ":";

showPoint(); //파생클래스에서 기본클래스멤버 접근

}

int main() {

ColorPoint cp;

ColorPoint\* pDer = &cp;

Point\* pBase = pDer; //업캐스팅

pDer->set(3, 4);

pBase->showPoint();

pDer = (ColorPoint\*)pBase; //다운캐스팅

pDer->setColor("red");

pDer->showColorPoint();

}

기본클래스 포인터가 가르키는 객체를 파생클래스의 포인터로 가리키는 것

명시적 타입변환 필요

Point객체를 가리키는 Point객체 포인터를 ColorPoint\*로 형변환하면 안됨

check time 1. 업캐스팅 – 3, 다운캐스팅 – 2

2. 4

8.4 protected 접근 지정

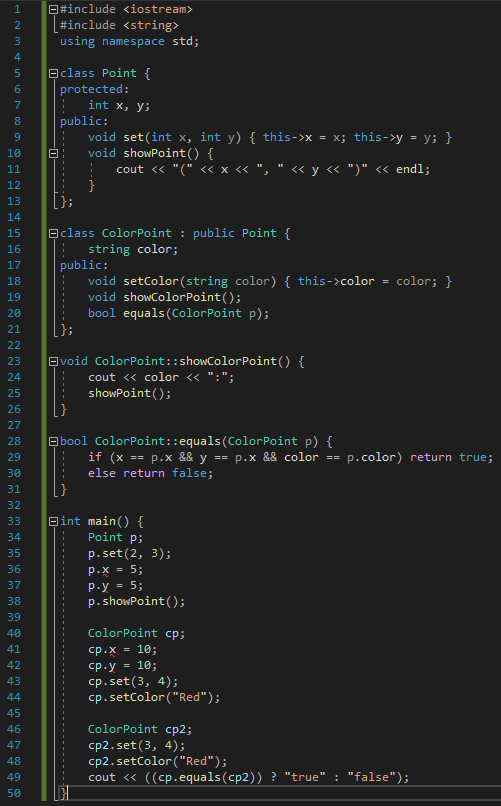
멤버의 접근 지정에 따른 접근 가능 여부 private protected public

멤버를 선언한 클래스 O O O

파생클래스 X O O

다른 클래스나 외부 함수 X X O

protected로 지정된 멤버 파생클래스에 접근 허용 but 다른 클래스, 외부함수에선 접근 불가

예제8-2 

8.5 상속과 생성자, 소멸자

파생클래스와 기본클래스의 생성자 호출 및 실행관계 기본클래스의 생성자 실행 🡪 파생클래스의 생성자 실행

컴파일러에 의해 호출관계 정리

소멸자의 실행 순서 파생클래스의 객체 소멸 파생클래스의 소멸자 실행 🡪 기본클래스의 소멸자 실행

생성자 실행 순서와 반대로 소멸자 실행

파생클래스에서 기본클래스 생성자 호출 파생클래스의 기본클래스 생성자 결정 파생클래스 생성자 작성 시, 기본클래스의 생성자를 지정해야 함

지정하지 않을시 컴파일러는 묵시적으로 기본클래스의 기본생성자 실행하도록 컴파일

컴파일러에 의해 묵시적으로 기본클래스의 생성자가 선택되는 경우 기본클래스에 기본생성자가 선언되지 않으면 컴파일 오류

매개변수있는 파생클래스 생성자를 실행해도 기본생성자 실행

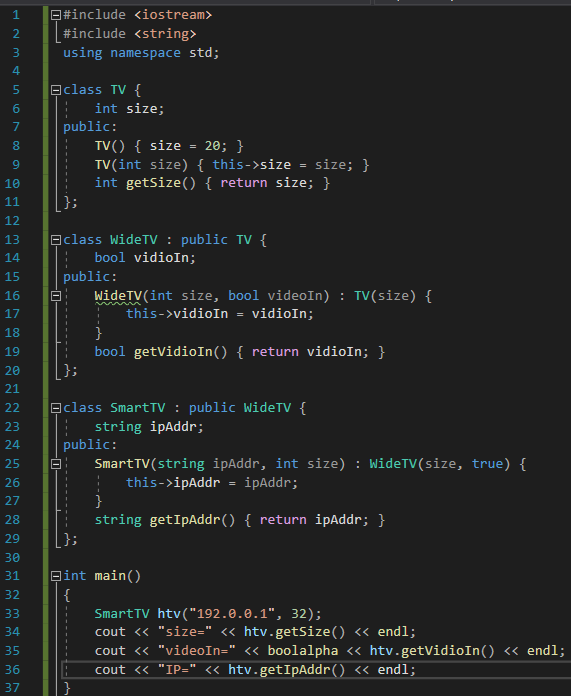
명시적인 기본클래스의 생성자 선택 파생클래스의 생성자 작성시, 기본클래스의 생성자를 지정해야 함

ex) B(int x) : A(x+3) { cout << "매개변수생성자 A" << x << endl; } //B b(5) --> A(8)호출

컴파일러의 기본 생성자 호출 코드 삽입 컴파일러의 명시적으로 A()를 호출하도록 B의 생성자 변형

ex) B() : A() { cout << "생성자 B" << endl; }

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

예제8-3 

check time 1. 8, 5, 30

2. 8, 4

8.6 상속의 종류: public, protected, private 상속

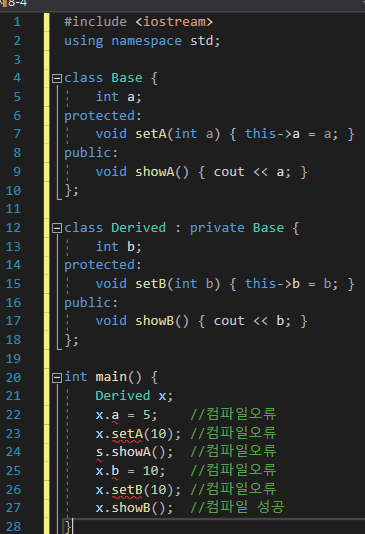
상속형식 class 파생클래스 : 접근지정자 부모클래스

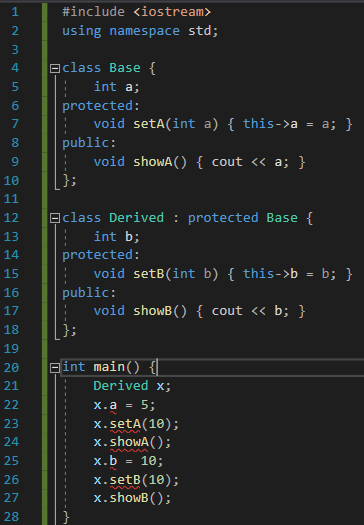
접근지정자 public 기본클래스의 protected, public멤버들이 접근 지정 변경없이 파생클래스에 상속

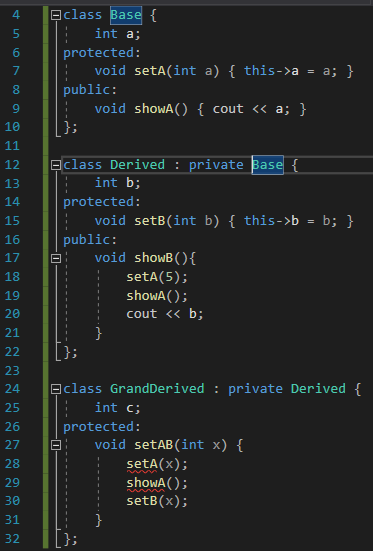
protected 기본클래스의 protected, public멤버들이 protected접근지정으로 변경되어 파생클래스에 상속

private 기본클래스의 protected, public멤버들이 private접근지정으로 변경되어 파생클래스에 상속

접근지정자 생략시 자동으로 private로 자동 처리

예제8-4 

예제8-5 

예제8-6 

8.7 다중 상속

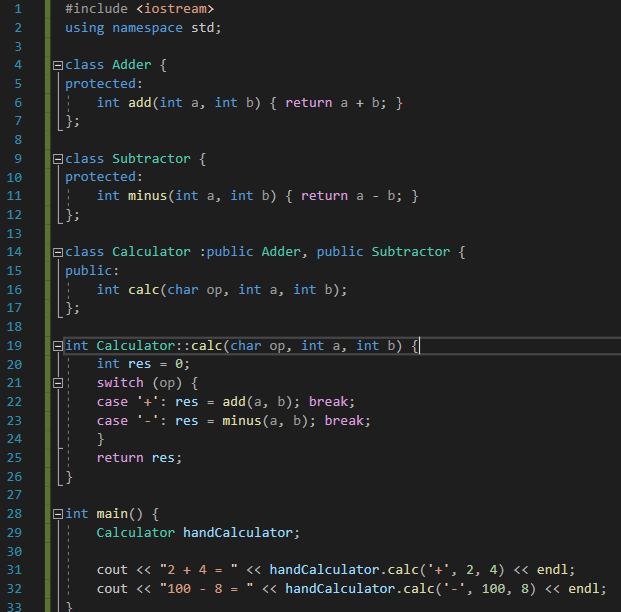
다중 상속 하나의 파생클래스가 여러 클래스를 동시에 상속받는 것

다중상속선언 클래스 선언문에서 접근지정자 기본클래스를 콤마로 나열

형식 class 파생클래스 : 접근지정자 기본클래스, 접근지정자 기본지정자

ex) class MusicPhone : public MP3, public MobilePhone {};

다중상속활용 다중상속받은 기본클래스들의 멤버들을 모두 호출 가능

예제8-7 

8.8 가상 상속

다중상속의 문제점 다이아몬드형 상속일시 모호함 발생 중복상속되어 문제발생

ex) class BaseIO {

public:

int mode;

};

class In :public BaseIO {

public:

int readPos;

};

class Out :public BaseIO {

public:

int writePos;

};

class InOut : public In, public Out {

public:

bool safe;

};

int main() {

InOut ioObj;

ioObj.readPos = 10;

ioObj.writePos = 20;

ioObj.safe = true;

ioObj.mode = 5; //모호함 발생, mode가 In의 기본클래스BaseIO의 mode인지, Out의 기본클래스BaseIO의 mode인지 모호

}

BaseIO가 중복 상속

가상상속 가상상속 선언 멤버중복 생성 문제 해결

형식 class 파생클래스 : virtual 접근지정자 기본클래스

ex) class In : virtual public BaseIO

표현 파생클래스는 가상기본클래스 기본클래스를 상속받는다

기본클래스를 가상상속받는다

가상상속으로 다중상속의 문제 해결 모호성 해결 멤버공간을 새로 생성하지 않고, 이미 할당된 공간을 공유

ex) class BaseIO {

public:

int mode;

};

class In : virtual public BaseIO { //가상상속, BaseIO의 멤버 상속

public:

int readPos;

};

class Out : virtual public BaseIO { //가상상속, BaseIO의 멤버 상속X --> 이미 BaseIO의 멤버가 상속되었기 때문

//In 선언 때 할당된 BaseIO 공간 공유

public:

int writePos;

};

class InOut : public In, public Out {

public:

bool safe;

};

int main() {

InOut ioObj;

ioObj.readPos = 10;

ioObj.writePos = 20;

ioObj.safe = true;

ioObj.mode = 5; //모호함 발생X, mode가 하나이기 때문

}

check time 1. (1) public : void getId() {retur id;}

1. (2) class Student : virtual public Person {};

class Worker : virtual public Person {};

2. 컴파일 타임