항목32 : public 상속 모형은 반드시 “is-a(~는~의 일종이다)”를 따르도록 만들자

public 상속일 때 **~는~의 일종이다. 즉, 파생객체는 기본객체의 일종이다.** 예를 들어보겠습니다.

class Person { . . . };

class Student : public Person { . . . };

모든 학생들은 사람이지만 모든 사람이 학생은 아닙니다. (학생객체는 사람객체의 일종이다.) c++에서 본다면 이렇게 말 할 수 있습니다. 사람타입의 인자를 받는 함수는 학생타입의 객체도 받을 수 있습니다. 학생도 사람이니까 당연한 결과 입니다. 코드로 보겠습니다.

*void eat ( const Person& p ); // 먹는 것은 사람이 할 수 있습니다.*

*void study ( const Student& s ); // 공부 하는 것은 학생이 할 수 있습니다.*

*Person p; // p는 사람 입니다.*

*Student s; // s는 학생 입니다.*

*eat ( p ); // 사람은 먹는 것을 할 수 있습니다.*

*eat ( s ); // 학생은 먹는 것을 할 수 있습니다. (학생도 사람이니까 먹을 수 있습니다.)*

*study( s ); // 학생은 공부를 할 수 있습니다.*

*study( p ); // 사람은 학생이 아니므로 공부할 수 없습니다.*

하지만 **“~는~의 일종이다.”가 맞지 않을 경우도 있습니다.** 예를 들어보겠습니다. “펭귄은 새의 일종이다.” 누구나 알고 있는 정의 입니다. 그리고 “새는 날 수 있다.” 이것 또한 사실 입니다. 그러면 “펭귄은 새의 일종이니까 펭귄은 날 수 있다.” 잘 나가다가 갑자기 이상한 길로 빠져 버립니다. 원인은 **“새는 날 수 있다.”에서 “모든 새는 날 수 있다”는 잘 못 된 가정이 포함 되어 있기 때문입니다.**

다음과 같이 더 현실에 가까운 클래스 계통구조를 뽑을 수 있을 것 입니다.

*class Bird { . . . // fly 함수가 선언되지 않았습니다. };*

*class FlyingBird: public Bird {*

*public:*

*virtual void fly( );*

*. . .*

*};*

*class Penguin: public Bird {*

*. . . // fly 함수가 선언되지 않았습니다.*

*};*

하지만 어떤 소프트웨어의 시스템의 경우엔 **비행 능력이 있고 없고를 구분 할 필요가 없는 경우에는 오히려 Bird를 기본 클래스로 하여 펭귄이 파생클래스가 되는 것이 더 좋은 설계입니다.**

*class Penguin: public Bird {*

*public:*

*virtual void fly( ) { error(“펭귄은 비행 할 수 없다.”); }*

*. . .*

*};*

위 코드가 나타내는 fly( )함수는 “펭귄은 날 수 없다”가 아닙니다. **“펭귄은 날 수 있다. 하지만 날려고 하면 에러가 난다”** 입니다.

**다음은 이상하고 아름다운 “public 상속” 세계를 보여드리겠습니다.**

모든 정사각형은 직사각형의 일종이다. 맞는 말입니다. 직사각형은 기본 클래스가 되고 정사각형을 직사각형의 파생 클래스가 됩니다. 코드로 나타내면 이렇습니다.

*class Rectangle {*

*public:*

*virtual void setHeight(int newHeight);*

*virtual void setWidth(int newWidth);*

*virtual int height( ) const; // 현재의 값을 반환합니다.*

*virtual int width( ) const;*

*. . .*

*};*

*void makeBigger(Rectangle& r) // 직사각형 객체의 가로길이를 늘리는 함수*

*{*

*int oldHeight = r.height( );*

*r.setWidth(r.width( ) + 10); // r의 가로 길이에 10을 더합니다.*

*assert(r.height( ) == oldHeight); // r의 세로 길이가 변하지 않는다는 조건에 단정문*

*}*

위 함수의 단정문이 실패할 일이 없습니다. 위 함수는 가로 길이만 바꾸고 있으니까요.

*class Square : public Rectangle // 직사각형을 상속받는 정사각형 클래스를 만듭니다.*

*{*

*. . .*

*}*

*Square s; // 정사각형 객체를 만듭니다.*

*assert(s.width( ) == s.height( )); // 이 단정문은 모든 정사각형에 대해 true여야 합니다.*

*makeBigger( s ); // 정사각형도 직사각형의 일종이므로 makeBigger함수를 호출 할 수 있습니다.*

*assert(s.width( ) == s.height( )); // 함수 호출 후에도 이 단정문은 모든 정사각형에 대해 true여야 합니다.*

그런데 함수 종료 후에 s의 가로길이가 변하여 **3번째 줄의 assert문에 걸려버립니다.** 원인은 이렇습니다. **정사각형은 포함 할 수 없는 직사각형만의 특징인 “가로 길이가 세로 길이에 상관없이 바뀔 수 있다”라는 부분입니다.** 그런데 우리에게 직면한 **문제는 public 상속은 기본 클래스 객체가 가진 모든 것들이 파생 클래스 객체에도 그대로 적용된다고 단정 합니다.** 이렇게 직사각형만의 특징이 있다는 사실을 알 리가 없는 컴파일러는 문법적 잘 못된 점이 없기 때문에 컴파일이 됩니다. 하지만 컴파일이 되었다고 해서 제대로 동작하는 것은 아닙니다.