항목 34: 인터페이스 상속과 구현 상속의 차이를 제대로 파악하고 구별하자

기본 클래스의 멤버 함수 인터페이스만을 상속 받고 싶을 때가 있습니다. 또는 인터페이스 및 구현 모두 상속받고 싶은 경우도 있습니다. 반대로 인터페이스 구현을 상속받되 오버라이드 할 수 없도록 막고 싶은 경우도 있습니다.

각각의 경우에 해당하는 방법을 설명 드리겠습니다.

* **기본 클래스의 멤버 함수 인터페이스만 상속 받고 싶은 경우**
* **순수 가상 함수**
* **인터페이스 및 구현 모두 상속 받고 싶은 경우**
* **단순(비순수) 가상 함수**
* **구현을 상속받되 오버라이드 할 수 없도록 막고 싶은 경우**
* **비가상 함수**

위 방법 대로 예제를 만들어 설명 드리겠습니다.

*class Shape { // 기본 클래스*

*public:*

*virtual void draw( ) const = 0; // 순수 가상 함수*

*virtual void error(const std::string& msg); // 단순 가상 함수*

*int objectID( ) const; // 비가상 함수*

*};*

*class Rectangle: public Shape { . . . }; // 파생 클래스*

*class Ellipse: public Shape { . . . }; // 파생 클래스*

Shape는 추상 클래스입니다. **순수 가상 함수인 draw가 Shape클래스를 추상 클래스로 만든 원인 입니다.** 추상 클래스는 인스턴스를 만들지 못하고 오직 파생 클래스만 인스턴스화가 가능 합니다.

기본 클래스 Shape의 각 멤버함수 선언에 대해 살펴 보도록 하겠습니다.

**먼저 순수 가상 함수인 draw입니다.** 순수 가상 함수는 파생 클래스에서 다시 선언하여 구현을 해야 합니다. 즉, 순수 가상 함수를 선언하는 목적은 파생 클래스에게 함수의 **인터페이스만을 물려주는 것 입니다.**

Shape::draw 함수는 그리는 용도 입니다. 타원을 그리는 것과 직사각형을 그리는 알고리즘은 서로 다릅니다. 따라서 당연히 각 파생 클래스마다 서로 알맞게 구현 해야 합니다.

**다음은 단순 가상 함수입니다.** 순수 가상 함수와는 다른 점은 파생 클래스에서 오버라이드할 수 있는 **함수 구현부도 제공한다는 점** 입니다. 같은 점은 인터페이스를 상속하는 점 입니다.

Shape::error 함수는 에러를 출력하는 역할을 합니다. 파생 클래스에서 이미 기본 클래스에서 정의된 에러 출력을 그대로 쓰고 싶으면 재정의 없이 사용하면 되고 어떤 파생 클래스에서는 특별하게 사용하고 싶으면 재정의 해서 써야 합니다.

**단순 가상 함수의 문제점은 파생클래스의 선택의 여지 없이 무조건 이 함수를 물려받아야 한다는 것입니다.** 어떤 파생클래스에서는 반드시 재정의 해서 사용해야 하는데 실수로 재정의 하지 않고 사용할 수도 있다는 것이죠.

*class Airplane{*

*public:*

*virtual void fly( )*

*{*

*// 날아가는 기본 동작*

*}*

*};*

*class ModelB: public Airplane { // 날아가는 동작이 특별한 모델*

*public:*

***. . . // 단순 가상함수를 재정의 하는 것을 잊었습니다!***

*}*

*ModelB::fly( ); // 에러! 모델B는 기본 클래스의 날아가는 방법을 쓸 수 없습니다.*

해결 방법은 간단 합니다. 가상 함수의 인터페이스와 구현의 연결 관계를 끊어 버리는 것입니다.  
*class Airplane { // 기본 클래스*

*public:*

*virtual void fly( const Airport & destination )* ***= 0;// 단순 가상에서 순수로 바꿉니다.***

*protected:*

***void defaultFly( const Airport & destination ); // 대부분의 파생 클래스가 사용합니다.***

*};*

*void Airplane::defaultFly( const Airport & destination );*

*{*

*// 비행기가 날아가는 기본 동작 원리를 가진 코드*

*}*

*class ModelA: public Airplane { // 날아가는 동작이* ***일반****적인 모델*

*public:*

*virtual void fly( const Airport & destination ) // 순수가상 함수 구현*

*{*

***defaultFly( destination ); // 기본클래스의 일반 동작 함수 호출***

*}*

*};*

*class ModelB: public Airplane {* ***// 날아가는 동작이 특별한 모델***

*public:*

*virtual void fly( const Airport & destination ) // 순수가상 함수 구현*

*{*

***// 날아가는 동작이 특별한 구현 정의***

*}*

*}*

이제 기본 클래스에서 **묻지마 물려주기를 막을 수 있습니다.** 순수 가상 함수를 구현하지 않는 이상 사용할 수 없기 때문입니다.

**마지막으로 멤버 함수가 비가상 함수로 되어 있는 경우 입니다.**

이 함수는 **파생 클래스에서 사용하든지 기본 클래스에서 사용하든지 상관없이 변하지 않는 동작을 지정하는 함수** 입니다. 미래에 만들어질 파생 클래스가 특수하더라도 이 함수는 변하지 않고 그대로 쓸 수 있는 것입니다.

Shape::objectID 함수는 어떤 파생 클래스든지 객체의 식별자를 반환하는 동작을 나타냅니다.