濟州大學校 産業技術硏究所 論文集, 第10卷, 2號, pp.91-98, 1999. 12
Cheju Nat'l Univ. Res. Insti. Ind.Tech. Jour., Vol.10, No.2 pp.91-98, 1999. 12

객체 자료의 입출력 설계에 대하여

변 상 용*

On a Input and Output Design for Object Data

Sang-Yong Byun*

ABSTRACT

There is a trend of increasing interest for software components reuse. Reuse based on objects is belived efficient than that on functions. Objects are implemented by classes, therefore we are focused on reuse of classes. This study is concerned with an increasing reusabiliy through separating input interfaces and output interfaces from the classs that have ultimate input data or ultimate output data.

Key words : Class reuse, input class data, output class data, interface separation

Ⅰ. 서 론

소프트웨어 개발 방법은 기능지향, 자료지향, 객체지향 접근방법으로 대별될 수 있다[1)2)]. 이 중에서 자료지향 접근방법은 그 적용영역이 한정[1)2)]되기 때문에 제외한다면, 기능지향과 객체지향 접근방법이 남게 된다. 기능지향 접근방법은 소프트웨어를 개발하던 초기부터 지금에 이르기까지 사용되고 있으며, 객체지향 접근방법은 80년대 초 에이다, 스몰토크 등의 객체지향 프로그램 작성 언어가 등장하면서 시작되었고, 지금은 기능지향 접근방법 보다 선호하는 접근방법이 되었다.

80년대 중반부터 일기 시작한 소프트웨어 부품의 재사용에 대한 관심은 지금은 물론 앞으로도 지속될 것이다[3)]. 소프트웨어 부품을 하드웨어의 칩과 같이 재사용하자는 소프트웨어 칩이라는 개념도 이때부터 등장하였다[4)]. 소프트웨어 부품의 재사용에 있어서, 기능지향 접근방법 보다는 객체지향 접근방법이 더 효과적이라는 것이 많은 학자들의 주장[5)]이고, 이 때문에 객체지향 접근방법이 많이 사용되고 있다.

네트워크 및 인터넷의 발달로 소프트웨어 개발방식도 개발자 개인이나 개발팀의 지역적 한계에서 벗어나, 나라 전체 및 세계전체에서 소프트웨어 부품을 구하여 소프트웨어 개발에 이용할 수 있다. 그리고 객체지향 프로그램 작성인 경우에는 MFC(Microsoft Foundation Class)[6)]와 같은 클래스 라이브러리의 발달로 재사용가능한 클래스가 상당히 많이 존재하고 있다. 또한 비쥬얼 C++나 비쥬얼 Basic 등에서는 API(Application Programming Interface)[7)]나, ActivX 컨트롤[7)]의 사용으로 인해, 소프트웨어 개발에 있어서의 개발자의 노력이 현저히 감소하고 있다.

이제는 한 개발자가 개발한 소프트웨어 부품은 개발자 한 사람의 자원이 아니라, 그가 속한 팀, 그 나

* 제주대학교 정보공학과, 산업기술연구소
Dept. of Information Eng., Res. Insti. Ind. Tech., Cheju Nat'l Univ.

라 및 세계적인 자원이 될 수 있다. 문제는 기존에 개발된 소프트웨어 부품에 관심이 있는 다른 사용자가 그 부품을 효과적으로 사용할 수 있는가이다.

본 논문에서는 객체지향 접근방법의 핵심인 객체가 가지는 자료 중 외부입력으로 그 값이 결정되는 것과 외부출력되는 것에 대한 입출력 방법에 따른 재사용성에 대하여 논한다.

II. 객체의 자료 및 연산

2.1. 궁극적 입출력 자료

객체는 Fig. 1 (a)와 같이 자료와 그 자료에 연관된 연산을 함께 가진다. 자료는 Fig. 1 (b)와 같이 외부적으로 공개하는 공적자료와 외부에는 감추는 사적자료로 나뉜다. 자료가 설계 의사결정 상 중요한 경우에는 사적자료로 다루며, 이외에도 관련이 없는 다른 객체가 임의로 자료값을 변경하는 것을 방지하기 위해, 자료는 주로 사적으로 만든다. 연산도 Fig. 1 (b)와 같이 외부에서 호출할 수 있는 공적연산과 외부에서는 호출할 수 없고, 내부적으로만 사용되는 사적연산으로 나뉘어진다. 또한 연산은 Fig. 1 (c)와 같이 객체자료 값을 변경시키는 변화자 연산과 자료 값에는 영향을 미치지 않으면서 객체외부로 값을 보내주는 접근자 연산으로 나뉜다. 연산을 변화자와 접근자 연산이라는 기준으로 나눈다면, 자료입력 연산은 변화자 연산으로, 자료출력 연산은 접근자 연산으로 볼 수 있다.

객체가 가지는 자료는 내부자료, 내부객체에 의해 입력되거나 내부객체로 출력되는 자료, 궁극적으로는 외부개체에 의해 입력되는 궁극적 입력자료와 궁극적으로는 외부개체로 출력되는 궁극적 출력자료로 나눌 수 있다. Fig. 2의 자료흐름도가 어떤 소프트웨어에 대한 1 단계 자료흐름도라고 한다면, 'a'와 'b'라는 자료는 직접 외부에서 입력받는 자료이고, 'i와 'j'는 직접 외부로 출력되는 자료이며, 나머지 여섯 개의 자료 중 대부분은 소프트웨어 내부 입출력 자료이다. 이와 같은 소프트웨어가 어떤 클래스들로 구현되는지에 상관없이 궁극적으로는 외부에서 입력받으며, 궁극적으로 외부로 출력되는 자료는 존재하는 것이다.

2.2. 객체 자료의 입출력

여기에서 관심의 대상은 이 소프트웨어를 객체지향 프로그램 작성 언어로 구현했을 때, 객체는 클래스에

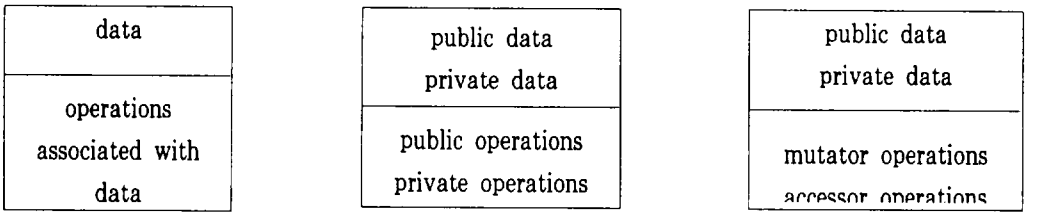


Fig. 1 Elements of object

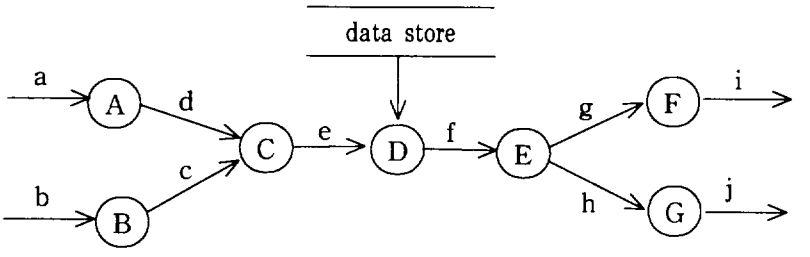


Fig. 2 Example Data Flow Diagram

의해 구현되는데, 궁극적인 입력자료를 가지는 클래스가 입력함수를 가지고 값을 설정하는가, 아니면 다른 곳에서 입력받아 변화자 함수에 의해 값이 설정되는가 하는 것과 궁극적인 출력자료를 가지는 클래스가 출력함수를 가지고 출력하는가, 아니면 다른 곳에서 접근자 함수에 의해 값을 전달받아 출력되는가 하는 것이다.

국내외의 객체지향 프로그램 작성에 대한 교재[8)-11)]를 살펴보면, 궁극적인 입력자료에 대해서는, Fig. 3과 같이 다른 곳에서 생성자 메소드를 호출하면서 매개변수를 전달하거나, 클래스가 가진 변화자 메소드에 자료를 전달하여 값을 할당하고 있다. 한편 출력자료에 대해서는, Fig. 4 (a)와 같이 클래스 자체의 출력 메소드에 의해서 출력하거나, Fig. 4 (b)와 같이 다른 곳에서 클래스의 접근자 함수를 사용하여 자료를 전달받아 출력하고 있다.

제주대학교 정보공학과 2학년 학생들의 1999년 제2학기 객체지향 프로그램 작성 교과과정에서, 아무런 언급없이 Fig. 5와 같은 문제만 제시하고 실습하도록 해보았다.

그 결과 29명중 27명이 궁극적인 입력자료에 대해

n 명의 학생들에 대한 학번, 중간, 기말, 출석, 과제 점수를 받아들여, 다음과 같은 출력을 내는 프로그램을 작성하시오.

*** 성 적 보 고 서 ***

학번 중간 기말 출석 과제 총점 학점

Fig. 5 Object-oriented programming example

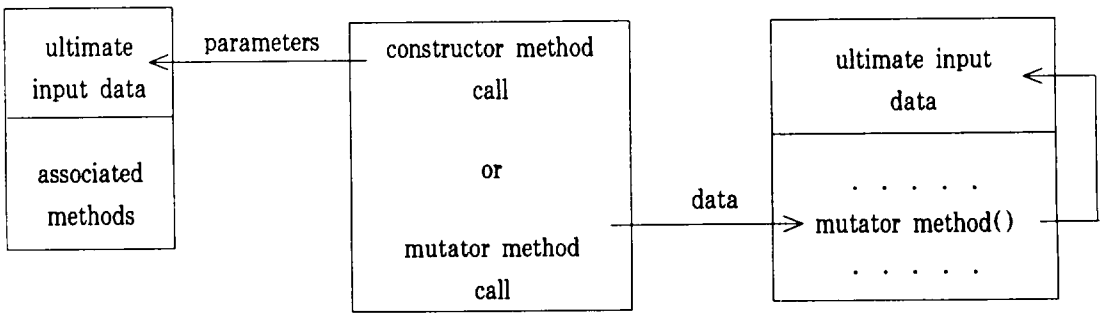


Fig. 3 Setting of ultimate input data

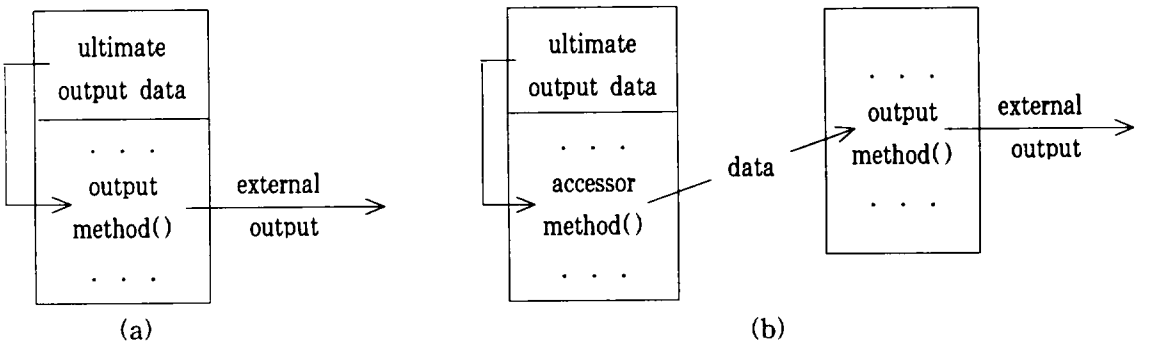
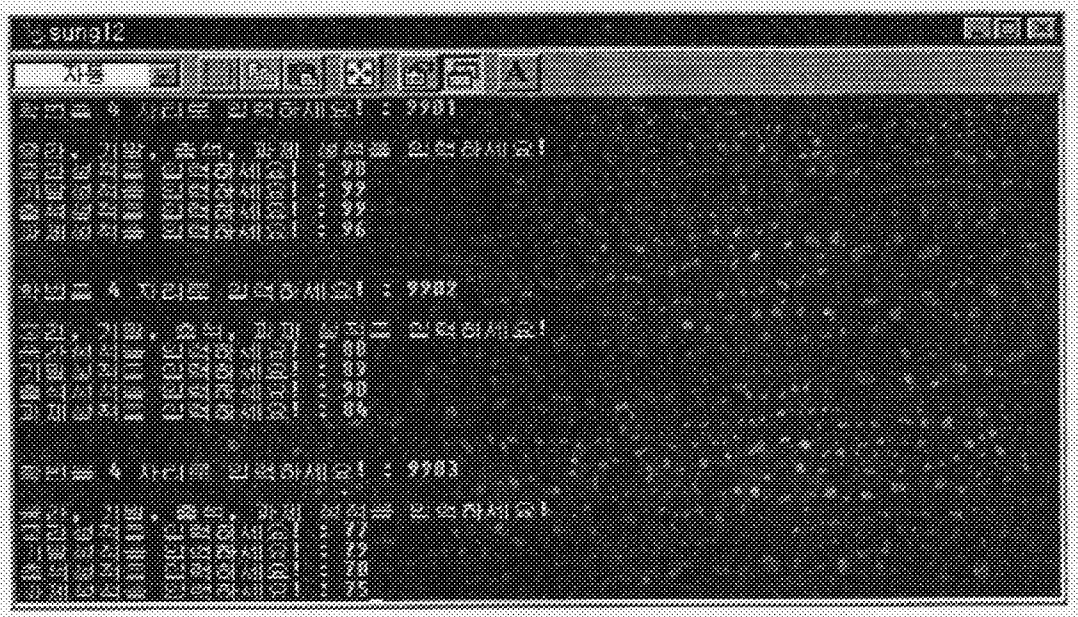
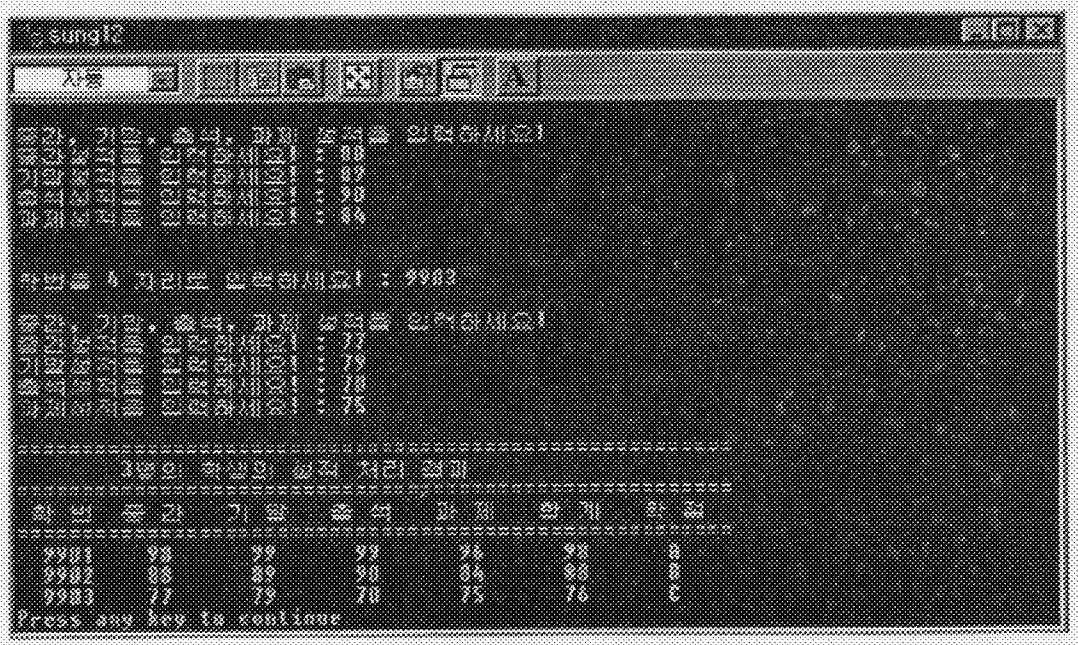


Fig. 4 Outputing of ultimate output data

서는 그 자료를 가지는 클래스에서 입력 메소드를 사용하여 입력했으며, 30명중 28명이 공식적인 출력자료에 대해서 그 자료를 가지는 클래스에서 출력 메소드를 사용하여 출력했다.



(a) Input process



(b) Output result

Fig. 6 Program excution example

2.3 객체 자료의 입출력 문제점

객체지향 프로그램 작성 교재에서 나타나며, 대부분의 학생들의 프로그램 작성에서 나타난 것처럼 프로그램을 작성했을 때, 소프트웨어 부품의 재사용에 심각한 영향을 미칠 수 있다. 즉 궁극적인 입력자료나 출력자료를 가지는 클래스가 직접 외부에서 입력받거나 외부로 출력하게 되면, 개발자 자신 또는 개발팀이 나중에 재사용할 때도 문제가 발생하지만, 개발자나 개발팀과 무관한 제 삼자가 그 클래스를 재사용할 때에는 더욱 심각한 문제가 발생한다. 개발자 자신이나 개발팀은 소프트웨어 부품에 대한 원시코드를 보존하는 한, 입력이나 출력에 있어서의 구현방법이 크게 바뀌어도, 입력 또는 출력 부분의 원시코드를 수정할 수 있어서 재사용 문제는 큰 어려움이 없을 수 있다. 하지만 제 삼자는 원시코드를 제공받지 않는 한, 입력이나 출력의 구현방법을 바꿀 수가 없기 때문에 재사용을 포기해야 하는 경우가 발생한다.

예를 들어, 객체지향 프로그램 작성에 대한 교재에서도 마찬가지이지만, 학생들이 프로그램을 작성한 문제를 살펴보자. 대부분의 학생들 처럼 구현했을 때, Fig. 6 (a)와 (b)는 터보 C++ 프로그램 작성 언어로 구현한 프로그램의 실행시의 입력과 출력 화면 예이다.

성적자료, 즉 학번, 중간, ... 학점(궁극적 입출력 자료)를 가지는 클래스가 자료에 대한 입출력 함수를 캡슐화하기 때문에, 성적처리 클래스를 사용하고자 하는 사용자는 반드시 Fig. 6 (a)와 (b)와 같은 인터페이스를 통해서만 자료를 입출력할 수 있다. 사용자가 원하는 형태로 자료를 입력하거나 출력시킬 수 없다. 사용자 인터페이스가 사용하기 편리하고, 사용자의 실수를 관대히 용서하며, 사용자가 선호하는 형태로 사용자와 상호작용해야 한다는 사용자 인터페이스 원칙 측면[12]에서 보면, Fig. 6과 같은 입력/출력 방법을 강요하는 것은 문제의 소지가 있다.

또한 사용자로 하여금 입력을 고객화할 수 있도록 해야 하며, 사용자가 선호하는 입력 모드로 조정될 수 있어야 하며, 사용자로 하여금 상호작용 흐름을 제어하도록 해야 한다는 입력 인터페이스 설계지침과, 사용자가 신속하게 이해할 수 있는 형태로 자료가 표현되어야 하며, 사용자가 시각적 문맥을 유지할 수 있도록 지원해야 한다는 출력 인터페이스 설계지침[12] 측면에서 살펴본다면, Fig. 6과 같은 인터페이스는 사용자에 따라 입력 및 출력 형태가 바꾸어야 할 만한 충분한 이유가 있다.

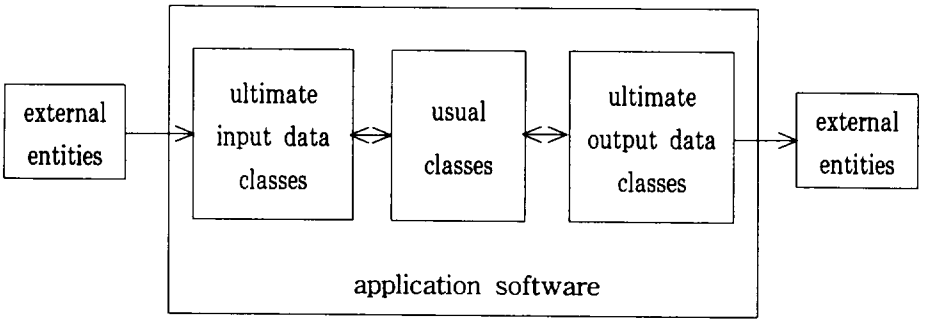
하지만 그러한 인터페이스에 대한 원시코드를 제공받지 못하는 사용자는 불만을 가지고 사용하던지, 아니면 기존의 클래스를 사용하지 않고 새로이 작성해야 한다. 이러한 것은 소프트웨어 부품의 재사용 측면에서 볼 때 바람직하지 못한 것이다.

III. 객체자료 입출력 설계의 분리

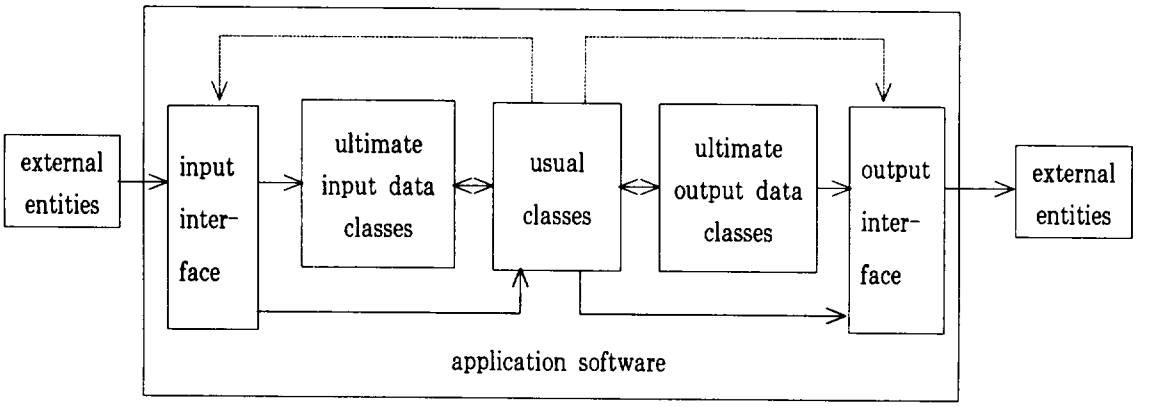
3.1 외부 인터페이스의 설계

사용자 인터페이스를 포함한 외부 인터페이스는 내부 모듈의 설계와 분리되어야 한다. 여기에서 말하는 외부 인터페이스는 응용 소프트웨어가 그 외부에 존재하는 개체인 인간, 하드웨어, 다른 소프트웨어 사이의 상호작용을 위한 인터페이스를 의미한다. Fig. 7 (a)와 같은 구조는 응용 소프트웨어에 존재하는 클래스들이 외부개체와 직접 연결된 반면에, (b)과 같은 구조는 응용 소프트웨어의 클래스들이 외부개체와 간접적으로 연결되었다.

Fig. 7 (a)에서, 외부개체는 응용 소프트웨어에 자료를 입력하거나 응용 소프트웨어에 의해 출력된 자료를 수령하는 인간, 하드웨어, 또는 소프트웨어를 의미한다. 궁극적 입력자료 클래스는 그 클래스가 가지는 자료중 적어도 하나는 외부개체가 입력하는 값에 의해 설정되어야 하는 클래스를 의미하며, 궁극적 출력자료 클래스는 그 클래스가 가지는 자료중 적어도 하나는 그 값을 외부개체로 출력해야 하는 클래스를 의미한다. 일반 클래스는 외부개체와는 전혀 상관없이 응용 소프트웨어가 담당한 작업을 성취하는 데에 필요한 클래스를 의미한다. 화살표는 자료가 흘러가는 방향을 의미한다. 응용 소프트웨어가 이러한 구조를 취한다면, 외부입력에 대한 제어는 전적으로 궁극적 입력자료 클래스가 가지며, 외부출력에 대해서는 전적으로 궁극적 출력자료 클래스가 가지며, 클래스의 사용자는 입출력에 대한 아무런 제어도 갖지 않게 된다.



(a) Direct connection between external entities and internal classes



(b) Indirect connection between external entities and internal classes

Fig. 7 Connection between external entities and internal classes

3.2 궁극적 입출력 클래스의 설계

Fig. 7 (b)에서는 입력 인터페이스와 출력 인터페이스들을 독립시켰다. 점선은 제어(여기에서는 호출)를 의미한다.

Fig. 7 (b)의 구조는 두 가지 특징을 나타낸다. 첫째, 궁극적 입력자료 클래스와 궁극적 출력자료 클래스는 외부개체와 직접 연결되지 않는다. 따라서 궁극적 입력자료 클래스가 입력 인터페이스를 호출하여 입력자료를 넘겨달라고 하지는 않는다. 만일 그와 같은 것을 허용한다면, 궁극적 입력자료 클래스가 입력에 대한 제어를 갖기 때문에 입력활동은 사용자의 제어를 벗어나게 된다. 또한 궁극적 출력자료 클래스가 출력 인터페이스를 호출하여 출력자료를 내보내라고 하지도 않는다. 만일 그와 같은 것을 허용한다면, 궁극적 출력자료 클래스가 출력에 대한 제어를 갖기 때문에 출력활동은 사용자의 제어를 벗어나게 된다.

둘째, 입력 인터페이스를 호출하는 클래스가 궁극적 입력자료를 가지는 클래스가 아니며, 출력 인터페이스를 호출하는 클래스가 궁극적 출력자료를 가지는 클래스가 아니라는 것이다. 일반 클래스에서 입력 인터페이스로 가는 점선 화살표는 외부개체와는 상관없는 일반 클래스들 중 하나, 즉 전체적인 제어를 담당하는 클래스에서 입력 인터페이스를 호출한다는 것을 의미한다. 이와 같은 호출명령은 일반적으로 사용자가 작성하는 메뉴 클래스의 함수나 주함수에 나타나도록 한다. 한편 일반 클래스에서 출력 인터페이스로 가는 점선 화살표는 외부개체와는 상관없는 일반 클래스들 중 하나, 즉 전체적인 제어를 담당하는 클래스에서 출력 인터페이스를 호출한다는 것을 의미한다. 이와 같은 호출명령도 일반적으로 사용자가 작성하는

메뉴 클래스의 함수나 주함수에 나타나도록 한다.

Ⅳ. 결과 고찰

본 논문에서는 궁극적 입력자료 클래스가 직접 입력을 요청하지 못하도록 하며, 궁극적 출력자료 클래스가 직접 출력을 요청하지 못하도록 제한하였다. 그렇게 함으로써, 그러한 클래스를 사용하려는 자는 자료의 입력 및 출력을 위한 코딩 작업을 행해야만 한다. 따라서 입력 메소드나 출력 메소드를 가지는 클래스를 사용하는 것에 비해 추가적인 작업으로 인한 비용과 노력이 낭비되는 것으로 생각할 수 있다.

하지만 사용하려는 클래스의 입력이나 출력 양식이 시대, 지역, 사람의 심리 등을 만족시킨다는 보장은 거의 불가능하다. 특히 사람은 저마다의 개성이 있어서, 개발자가 주의를 기울렸다고 해도, 상호작용 방식에 불만을 가지는 집단은 존재하기 마련이다. 소프트웨어는 말 그대로 소프트하기 때문에, 시대, 지역, 사람의 상황에 유연하게 대처할 수 있어야 한다.

또한 외부개체에 속하는 하드웨어나 다른 소프트웨어는 클래스를 개발할 당시의 가정과 같지 않을 수가 있다. 한 하드웨어를 가정한 입력이나 출력을 가지는 클래스는 다른 구조를 가지는 하드웨어에서 재사용하기 어렵다. 예를 들어 아이비엠 호환기종을 염두에 둔 클래스는 맥킨토시와 같은 하드웨어에서는 그 클래스를 사용하는 것이 어려울 수 있다. 또한 한 운영체제를 가정한 입력이나 출력을 가지는 클래스는 다른 운영체제를 사용하는 환경에서는 사용할 수 없을 가능성이 많다. 예를 들어, 윈도우즈에서 사용될 것이라고 가정해서 만든 클래스를 도스에서 사용하기는 어렵다.

따라서 객체가 가지는 자료에 대한 입력과 출력과 같은 외부개체와의 상호작용 작업은 그렇지 않은 작업과 분리시켜야 하는데, 본 논문이 그러한 측면을 추구했다.

Ⅴ. 결 론

본 논문은 객체가 가질 수 있는 자료중 외부개체와 연관되는 자료의 입력과 출력에 대해서 논하였다.

외부개체와 연관되는 자료 입력과 출력에 대한 설계를, 내부모듈에 대한 설계와 함께 고려하기보다는, 분리시킴으로써 클래스의 재사용을 증진시킬 수 있는 측면을 살펴보았다. 즉 본 논문에서 논한 방식으로 클래스를 개발한다면, 원시코드를 획득할 수 없는 사용자라 하더라도 클래스를 재사용할 수 있는 장점이 있다. 또한 외부개체의 특성에 독립적인 재사용을 지원하는 특징도 있다.

소프트웨어 개발 대상영역은 넓으며, 재사용 대상 또한 많다. 소프트웨어 부품의 재사용을 통해서 소프트웨어 개발 비용과 노력을 감소시킬 수 있고, 개발된 소프트웨어의 품질을 보증할 수 있는한 소프트웨어 개발자의 기존 클래스에 대한 재사용 수요는 더욱 늘어날 것이다. 현재뿐만 아니라 네크워크와 인터네트가 더욱 발전되는 미래에, 원하는 클래스를 지금보다는 더욱 용이하게 구할 수 있는 환경을 갖추게 될 것이다. 이런 상황에서 원시코드를 획득하지 않고도, 재사용할 수 있는 클래스를 개발한다는 것은 중요한 것이며, 본 논문이 그러한 방향에서 일조를 한다.

참고문헌

1) Michael J. Pont, 1996, *Software Engineering with C++ and CASE Tools*, Addison-Wesley
2) Roger S. Pressman, 1998, *Software Engineering-A Practitioner's Approach 4th ed.*, McGrow-Hill
3) Frakes,W. and Isoda,S. 1994. *Sucess factors for systematic reuse*, IEEE Software, 11(5), 14-22
4) Matsumoto,Y., 1984, *Some experience in promoting reusable software: presentation in higher abstract levels*, IEEE. Trans. on Software Engineering, SE-10(5), 502-12
5) Gamma,E., Helm,R., and Vlissides,J., 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley
6) 김용성, 1999, Visual C++ 6 완벽 가이드, 영진출판사
7) Jeffrey P. Mcmanus, 1997, *HOW to Program Visual Basic 5.0 Control Creation Edition*,

Ziff-Davis Press
8) Walter Savitch, 1996, *Problem Solving with C++, the Object of Programming*, Addison-Wesley
9) Jesse Liberty, 1999, *SAMS Teach Yourself C++ 3rd ed.*, SAMS
10) 천금문, 1998, C++ 프로그래밍, 정일 출판사
11) 백영승, C++ 기초에서 활용까지, 1999, 박문각
12) Dumas, J.S., *Designing User Interface for Software*, Prentice-Hall, 1988