

석사학위논문

객체지향 프로그래밍 기초능력 배양을
위한 시각적 도구의 이용 방안

지도교수 김 철 민



제주대학교 교육대학원

컴퓨터교육전공

양 정 민


2001년 8월

객체지향 프로그래밍 기초능력 배양을 위한 시각적 도구의 이용 방안

지도교수 김 철 민

이 논문을 교육학 석사학위논문으로 제출함

2001년 4월 일

 제주대학교 중앙도서관
JEJU NATIONAL UNIVERSITY LIBRARY
제주대학교 교육대학원 컴퓨터교육전공

제출자 양 정 민

양정민의 교육학 석사학위논문을 인준함

2001년 7월 일

심사위원장 _____ 인

심사위원 _____ 인

심사위원 _____ 인

<국문초록>

객체지향 프로그래밍 기초능력 배양을 위한 시각적 도구의 이용 방안

양 정 민

제주대학교 교육대학원 컴퓨터교육전공

지도교수 김 철 민

시각화는 학습에 대한 집중도를 높여 학습자들이 학습 목표를 더 빠르고 깊이 있게 이해하도록 지원하는 유용한 교수·학습 수단이다. 컴퓨터 기술이 발전하면서 다양한 분야에서 시각화를 통한 교수·학습 방법이 연구되고 있으며, 소프트웨어 개발 및 시스템 설계 수요가 높아짐에 따라 그 중요성이 커지고 있는 객체지향 프로그래밍 분야 역시 관련 개념들을 시각화하기 위한 연구가 활발히 진행되고 있다. 이들 연구의 결과로 몇몇 시각적 도구들이 개발되었지만 그들의 기능 및 특성에 대한 비교·분석이나 현실적 이용 방안 등이 제시되지 않아 그 활용도는 그리 높지 않다.

본 논문은 객체지향 프로그래밍 교육을 위해 개발된 기존의 시각적 도구 몇 가지를 선정하여 그들의 특성을 분석하고, 객체지향 프로그래밍에 대한 기초능력을 배양하기 위한 이용 방안을 제시하고 있다. 이를 위해 진행된 연구 내용을 정리하면 다음과 같다.

첫째, 객체지향 프로그래밍과 관련된 개괄적 내용으로서, 그 도입 배경과 특징, 장·단점, 객체지향 프로그래밍 언어의 종류 등을 기존 자료들을 조사하여 정리하였다.

둘째, 객체지향 프로그래밍과 관련된 내용을 교수·학습할 때 다루어야 할 내용으로 체계적으로 정리하였다. 먼저 객체지향 프로그래밍 교육의 필요성을 기술하고, 교육 대상으로서 객체지향 프로그래밍 언어인 자바를 선택한 배경을 설명하였다. 특정 언어에 국한된 개념이나 프로그램 수행 환경에 대한 잘못된 인식으로 인해 교수자나 학습자가 부딪힐 수 있는 교수·학습의 어려움을 지적하였고, 객체지향 프로그래밍과 관련된 교수·학습의 주요 내용을 객체와 클래스, 객체 간 상호작용, 클래스 간 관계의 세 가지 영역으로 구분하여 정리하였다.

셋째, 객체와 클래스, 객체 간 상호작용, 클래스 간 관계를 중심으로 객체지향 프로그래밍 교육을 위한 시각적 도구들의 시각화 방법을 비교·분석함으로써, 프로그래밍 기초능력을 체계적으로 배양하기 위해 적용할 수 있는 기존 도구들에 대한 상호 보완적 이용 방안을 도출하였다. 이에 덧붙여 객체지향 프로그래밍 기초능력 배양을 위해 시각적 도구가 충족시켜야 할 주요 조건으로서, 시각화 영역 선정의 체계성, 사용자와의 상호작용성, 선택적/수준별 학습 가능성 등을 제시하고 있다.

* 본 논문은 2001년 8월 제주대학교 교육대학원 위원회에 제출된 교육학 석사학위 논문임.

목 차

초 록	i
그 립 차 례	iv
I. 서 론	1
1. 연구의 필요성과 목적	1
2. 연구 내용 및 방법	3
II. 이론적 배경	4
1. 객체지향 프로그래밍의 배경	4
2. 객체지향 프로그래밍 언어의 종류	4
3. 객체지향 프로그래밍의 특징	5
4. 객체지향 프로그래밍의 장점과 단점	6
III. 객체지향 프로그래밍 교수·학습	8
1. 객체지향 프로그래밍 교수·학습의 필요성	8
2. 객체지향 프로그래밍 교육용 언어의 선택	9
3. 객체지향 교육의 유의점	12
4. 객체지향 프로그래밍 교수·학습의 내용	13
4.1. 객체와 클래스	17
4.1.1. 객체	17
4.1.2. 클래스	18
4.1.3. 클래스와 객체의 관계	19

4.2. 객체 간 상호작용	21
4.2.1. 객체 간 관계	21
4.2.2. 객체의 상태변화	22
4.3. 클래스 간 관계	22
4.3.1. 상속	22
4.3.2. 다형성	24
IV. 시각적 도구의 이용 방안	25
1. 도구의 특징	25
2. 시각화 방법의 비교·분석	26
2.1. 객체와 클래스의 시각화	26
2.1.1. 객체 구성의 시각화	26
2.1.2. 클래스 구성의 시각화	29
2.1.3. 클래스와 객체의 관계 시각화	31
2.2. 객체 간 상호작용의 시각화	32
2.3. 클래스 간 관계의 시각화	36
3. 시각적 객체지향 프로그래밍 도구의 이용 방안	38
V. 결론 및 제언	41
참 고 문 헌	43
ABSTRACT	45

그림 차례

[그림1] 객체지향 프로그래밍	16
[그림2-a] 객체의 구성	27
[그림2-b] 인스턴스 메소드	27
[그림2-c] 인스턴스 변수	27
[그림3-a] 객체 구성 초기화	28
[그림3-b] 객체 변수 이름 할당	28
[그림3-c] 객체 구성 완료	28
[그림4-a] 클래스 구성	29
[그림4-b] 클래스 편집 창	29
[그림5-a] 코드의 실행과 관련 클래스	30
[그림5-b] 클래스 코드 대체	30
[그림6-a] 클래스와 객체의 관계	31
[그림7-a] 클래스와 객체의 관계	32
[그림8-a] 객체의 상태 변화 A	33
[그림8-b] 객체의 상태 변화 B	33
[그림8-c] 객체의 상태 변화 C	34
[그림8-d] 객체의 메소드 호출 & 결과	34
[그림9-a] 객체 간 참조관계와 상태 변화 A	35
[그림9-b] 객체 간 참조관계와 상태 변화 B	35
[그림9-c] 객체 간 호출관계	36
[그림10-a] 클래스 간 참조관계	37
[그림10-b] 클래스 간 상속관계	37
[그림11-a] 클래스 간 참조관계	38

I. 서론

1. 연구의 필요성과 목적

현재 객체지향 방법은 소프트웨어 개발과 시스템 설계의 거의 모든 분야에서 사용되고 있어 이에 대한 교육적인 관심이 크다. 객체지향 방법은 객체지향 프로그래밍 언어를 이용하여 객체지향 프로그램으로 구현되고, 객체지향 프로그래밍은 컴퓨터 프로그램을 바라보는 시각을 객체들의 상호작용으로 개념화시킨 것을 말한다.

객체지향 프로그래밍은 프로그래밍 경험이 있던 학습자를 포함한 객체지향 프로그래밍 초보자들이 학습하기에 어려운 것이며 이에 대한 하나의 방안으로 시각적인 교수·학습 자료와 교수 기술이 고려된다[Raner00]. 객체지향 프로그래밍 입문 단계를 강의하는 국내외 컴퓨터 관련학과에서도 어떻게 하면 쉽게 객체지향 프로그래밍을 교수·학습할 것인가에 대해 많은 연구를 하고 있고, 그 중 한 분야가 객체지향 프로그래밍 개념을 시각적으로 전달하고자 하는 시각적인 객체지향 프로그래밍 도구의 개발이다.

Bruner는 기본 개념이나 원리의 상호 관련성을 지식의 구조로 보고 이를 가르치는 방법으로 발견 학습의 중요성을 강조하였다. 학생들은 관찰과 실험으로 스스로 새로운 개념을 발견해 가는 학습을 통해서 문제 해결력을 터득하게 되고, 새로운 상황에서 학습한 개념을 적용할 수 있는 능력을 형성한다고 하였다. 기본 개념의 성격을 동일하게 유지하면서 학습자의 학습 단계에 적합한 형태로 제시하면 학자들이 하는 지적 활동을 초등 학생들도 할 수 있다고 주장하였던 바와 같이 객체지향 프로그래밍 초보자들에게도 객체지향 프로그래밍에 대한 기초 개념을 어떻게 구조화하여 제시하느냐

하는 것은 교수·학습의 성공을 결정하는 중요한 문제라고 볼 수 있다.

전통적으로 언어 교육 방식에는 각 분야에서 완성된 원칙을 먼저 배우고 이를 확인하거나 응용하는 연역적 교육 방식과 실제 문제 해결 과정을 통하여 지식체계를 형성하는 귀납적 방식이 있다. 놀라운 속도로 발전하고 있는 컴퓨터 분야에서 체계화된 지식을 하나씩 섭렵하기에는 너무 늦을 뿐 아니라 이미 체계화된 지식은 시간이 지남에 따라 유용성도 떨어진다. 이에 따라 지식 자체보다는 이를 발전시키는 능력인 문제 해결력에 초점을 둔 귀납적 방식으로 프로그래밍을 교수·학습해야 한다고 주장하고 있다 [표창우98].

본 연구의 목적은 이와 같은 귀납적 프로그래밍의 일환으로 시각적 도구를 이용하여 객체지향 프로그래밍 기초 개념을 학습하기 위한 방안을 탐색하고자 하는 데 있다. 시각화는 컴퓨터 교육의 많은 분야에서 유용한 도구이며, 객체지향 프로그래밍 교육에 있어서도 객체지향의 핵심 개념들을 직관적으로 시각화하기 위한 연구들이 진행되고 있다. 더구나 객체지향 프로그래밍 학습에서는 특히 객체지향 프로그래밍의 객체와 클래스와 같은 기초 개념에 대한 확실한 이해가 중요하며 이와 관련하여 객체지향 프로그래밍의 기초능력을 배양하기 위한 교수·학습 방법의 하나로 시각적 도구들을 비교·분석하여 그 이용 방안을 제안하였다.

이를 위해 II장에서는 객체지향 프로그래밍의 배경에 대해서 살펴보았고 이어서 III장에서는 객체지향 프로그래밍 교수·학습에 대한 필요성, 객체지향 프로그래밍 교육에서의 유의점과 교수·학습 내용과 관련하여 다루었다. 그리고 IV장에서는 객체와 클래스, 객체 간 상호 작용, 클래스 간 관계가 시각적 도구에 어떻게 나타내고 있는 지 비교·분석함으로써 시각적 도구의 이용 방안을 탐색하였다. 마지막으로 V장에서는 앞장에서의 논의를 바탕으로 객체지향 프로그래밍 학습을 지원하기 위한 시각적 도구는 어떤

방향으로 개발되어야 할 것인지에 대해 제안하였다.

2. 연구 내용 및 방법

- 객체지향 프로그래밍에 대한 이론적 배경과 객체지향 프로그래밍 교수·학습에서의 유의점을 알아보았다.
- 객체와 클래스, 객체 간 상호작용, 클래스 간 관계로 구분하여 객체지향 프로그래밍 관련 내용을 정리하였다.
- 객체지향 프로그래밍 개념이 시각적 프로그래밍 학습 도구에 어떻게 나타나고 있는지 시각적인 특징을 그림으로 캡춰하여 비교·분석하고 이를 바탕으로 시각적 도구의 이용 방안을 추출하였다. 분석대상은 시각적인 도구인 1)BlueJ와 Monash 대학에서 Toolbook으로 제작된 자바 데모 자료(이하 2)Java Demonstrations라고 명명함)이다. 이에 덧붙여 Object Visualizer[Dershem&Vanderhde98]의 유용성을 검토한다.

본 논문에서 분석 대상이 된 시각적 도구는 특정한 객체지향 프로그래밍 언어와 관련된 도구로서 객체지향 프로그래밍에 대해서 배우는 것을 특정 언어를 배우는 것으로 잘못 인식할 수 있다. 따라서 일반적으로 적용될 수 있는 도구의 개발이 필요하다. 또한 본 논문에서 제시한 객체지향 프로그래밍 학습을 위한 시각적 도구의 이용 방안이 실제 현장에 적용될 수 있도록 구체적인 사례를 만들고 실시해 보는 추가적인 연구가 필요하다.

1) BlueJ는 <http://www.bluej.org>에 소개됨

2) Java Demonstrations는 <http://www.pscit.monash.edu.au/~ainslie/cpe1001/files.htm>에 소개됨

II. 이론적 배경

1. 객체지향 프로그래밍의 배경

객체지향 프로그래밍은 Simula[67]에 그 뿌리를 두고 있다. Simula는 1967년 노르웨이에서 개발된 최초의 객체지향 프로그래밍 언어로 클래스를 지원한다. 이것은 실세계에서 일어난 일을 컴퓨터로 시뮬레이션하기 위해 만들어진 모의시뮬용 언어로 Smalltalk와 Flavors에 영향을 끼쳤다. Smalltalk는 '객체지향'과 객체지향 프로그래밍을 널리 보급하는데 가장 많은 공헌을 한 언어로 1980년대 초반에 발표되었다. Smalltalk의 설계 철학의 근간은 소프트웨어 분야에서 프로그램이 점차 대형화되면서 기존의 방법에 한계가 도출되는 소프트웨어 위기를 극복하고자 하는데 있었다. Flavors는 LISP를 중심으로 발전되어온 것으로 주로 인공지능 분야에서 사용되어온 객체지향 언어이다[Madsen&Møller-Perderson88].

2. 객체지향 프로그래밍 언어의 종류

객체지향 프로그래밍 언어는 위에서 소개한 Simula, Smalltalk 외에도 C++, Eiffel, Objective-C, CLOS와 같은 언어들과 최근에 많은 관심을 끌고 있는 Java가 있다. 이들 언어의 특징을 살펴보면 다음과 같다.

C++은 C언어를 기반으로 만들어진 객체지향 언어로 AT&T의 Stroustrup에 의해 개발되었고 다중 상속을 지원한다. CLOS는 특정 연구 분야에 국한되지 않고 다양한 응용에 적합한 Common Lisp의 확장이다.

Java는 1995년 미국 샌프란시스코에 열린 SunWorld'95에서 공식 발표된 것으로 처음에 작은 가전제품을 만드는 것이 목적이었으나 모자익과 네스케이프가 인터넷을 보다 대중적으로 만들고, 클라이언트 서버 환경과 분산 시스템 환경이 무르익게 되자 Java의 구현을 인터넷으로 돌리게 되었다. C++와는 달리 Java에서는 클래스 외부에서는 변수나 메소드 등을 선언 할 수 없고, 모두 클래스 내에서만 선언되고 구현되어 사용될 수 있다.

Eiffel은 B. Meyer에 의해 개발된 언어로 다중 속성이 지원된다. Eiffel 컴파일러는 플랫폼 사이의 호환성을 위해 C로 번역한다. Objective-C는 Cox에 의해 개발되고, Next computer에 이용된 언어로 동적인 기억장소의 명시적인 관리인 할당과 해제가 요구된다[우치수98].

3. 객체지향 프로그래밍의 특징

객체기반 프로그래밍(object-based programming)은 객체와 클래스의 개념을 기반으로 객체 간에 상호작용 하는 것이며 객체지향 프로그래밍(object-oriented programming)은 객체와 클래스 외에 클래스를 통해 또 다른 클래스를 정의하는 상속 및 다형성이 추가된 것이다. 이 때 상속은 객체지향 프로그래밍과 객체기반 프로그램을 구분 짓는 중요한 메카니즘이다[Allen00].

객체지향 프로그래밍은 시스템의 기능에 초점을 맞추는 대신 그 시스템에 관여된 실세계의 물리적 모델을 만들려고 하는 것이다. 이 모델은 그 시스템이 가질 수도 있는 다른 기능들에 대한 기초를 형성하며 기능들은 나중에 변화될 수도 있고, 기본적인 모델의 변화 없이도 새로운 기능이 추가될 수도 있다. 객체지향 프로그램은 시스템의 기능보다 더 안정된 물리적 모델로 프로그래밍 원형이라고 볼 수 있다[Madsen&Møller-Perderson88].

4. 객체지향 프로그래밍의 장점과 단점

객체지향 프로그래밍의 장점은 우선 실세계의 현실을 반영하는 자연적 모델링이라는 데에 있다. 자연적 모델링이라는 것은 객체 및 클래스와 상속 구조가 문제영역의 정보를 사람들의 사고 방식과 매우 유사한 방법으로 찾고 표현하게 하는 것을 말한다. 즉 객체지향 프로그래밍의 바탕이 되고 있는 기본적인 철학은 가능한 현실세계의 일부를 반영하는 프로그래밍을 통하여 프로그램에서 설명되고 있는 것을 더 쉽게 이해하고 검토하게 하는 것이다.

또 한 가지 객체지향 프로그래밍의 중요한 장점은 코드의 재사용이다. 프로그램 구성요소들을 프로그래머들이 반복적으로 재구현하기보다는 재사용할 수 있도록 하는 것이다. 재사용은 이용하고자 하는 구현이 이미 있을 경우에는 단순한 문제이나 필요로 하는 객체와 정확히 일치되는 구현이 없을 때는 가장 비슷한 기존의 객체를 이용한다. 객체지향 프로그래밍은 이 목적을 위하여 포괄적인 코드의 사용을 제공한다. 포괄적인 코드란 코드를 일반적으로 써서 어떤 타입에도 돌아가도록 하는 것으로서, 이것을 예를 들어 설명하면 객체의 배열들을 정렬하기 위하여 이용되는 로직(logic)은 정렬되고 있는 객체들의 타입과 독립적이 되도록 하는 것이다[Allen00].

이에 비해 객체지향 프로그래밍의 단점은 너무 많은 작은 객체들이 만들어지고 쓰레기 수집기(garbage collector)에 의해 제거되어야 하는 비효율적인 특징이 있다. 쓰레기 수집기는 자동으로 수행되지만 이 과정도 작업이 필요하므로 많은 개수의 쓰레기를 수집하는 경우 실행시간이 극히 중요한 응용프로그램의 수행을 방해할 수 있다. 쓰레기 수집이 발생하는 정확한 시기나 발생 여부를 자체 예측하는 것이 매우 어렵기 때문이다. 이러한 문제는 실시간 소프트웨어와 같이 실행시간이나 기억 장소 요구에 대한 심

한 제약을 가진 시스템 개발에 장애로 작용한다[Madsen&Møller-Perderson88]. 그런데 Smalltalk가 정수형과 같은 기본 데이터 유형을 객체로 취급하여 수행 시간을 많이 소비하는데 반해 Java는 정수형(integer)뿐만 아니라 문자형(character)과 문자열(string)을 객체가 아닌 기본 데이터 유형으로 다루어 객체지향 프로그래밍의 순수성보다는 성능을 중요시하는 실용성을 택했다[Armstrong99].

또 한 가지 객체지향 프로그래밍의 단점은 메소드 매개변수에 따른 실행 시간 체크와 이름 바인딩이 느리고 불안정한 시스템을 만든다는 것이다. 그러나 이와 같은 문제는 오늘날 빨라진 컴퓨터의 속도로 극복되고 있으며 애플리케이션은 명령을 실행하는 시간보다 오히려 사용자 입력을 기다리는데 더 많은 시간이 걸린다.



Ⅲ. 객체지향 프로그래밍 교수 · 학습

1. 객체지향 프로그래밍 교수 · 학습의 필요성

Pratt와 Zekowitz는 프로그래밍 언어를 공부해야 하는 이유를 다음과 같이 제시하고 있다. 첫째 현재 이용하고 있는 프로그래밍 언어의 이용을 증진시키고, 둘째 여러 언어에서 제공하는 기능과 그 구현 방안을 익힘으로써 프로그래머의 어휘력을 향상시키며, 셋째 여러 프로그래밍 언어의 장점과 단점을 배움으로써 적절한 시기에 적합한 언어를 선택할 수 있고, 넷째 여러 프로그래밍 언어의 기능과 구현 기술을 알고 있으면 새로운 언어들의 습득이 용이해진다[우치수98]. 이를 고려하여 객체지향 프로그래밍을 교수 · 학습해야 할 필요성을 살펴보면 다음 몇 가지를 들 수 있다.

첫째, 현재 우리가 사용하고 있는 대부분의 응용 프로그램은 객체지향 프로그래밍 언어의 형태와 유사한 사용자 인터페이스를 가지고 있다. 학습자들이 자신의 컴퓨터에서 사용하는 일반 소프트웨어와 프로그래밍 시간에 배우는 작고 사소한 프로그램과의 차이가 너무 크다는 것은 학습자들이 프로그래밍 학습에 대한 흥미를 잃게 할 수도 있다. 따라서 학습자가 사용하고 있는 응용 프로그램과 관련된 프로그래밍을 고려할 필요가 있다.

둘째, 앞으로 개발 · 활용될 많은 실용적인 언어들이 어떤 모습으로든 객체 개념을 가질 것으로 예상되기 때문에 프로그래밍 교육에서는 객체지향 프로그래밍이 어렵다고 기피할 일만은 아니다.

셋째, 위에서도 지적한 바와 같이 일반적으로 사용되고 있는 응용프로그램에 사용자 인터페이스의 문제점이 발생할 경우 객체지향 프로그래밍 언

어의 각종 구성 요소와 구현 방안을 잘 아는 프로그래머가 이러한 문제를 쉽게 해결할 수 있다.

넷째, 최근 컴퓨터 프로그래밍 패러다임이 절차적 프로그래밍 패러다임에서 객체지향 패러다임으로 변화하고 있고 이와 같은 사회적·시대적 요구에 뒤처지지 않기 위해서 컴퓨터 교육과정 중 핵심 영역인 프로그래밍 교육에 객체지향 프로그래밍이 교육되어야 한다. 이제는 객체지향 프로그래밍을 하지 못하면 진정한 프로그래머라고 말할 수 없다.

다섯째, 객체지향 프로그래밍을 하기 위해서는 객체지향 개념들과 문제 상황과 객체지향 언어가 서로 연결되어 있어야 하는데 학습자들은 이 과정을 반복하게 됨으로써 교육의 전 분야에서 강조되고 있는 문제 해결력을 기를 수 있다[Hadjerrouit99].



2. 객체지향 프로그래밍 교육용 언어의 선택

최근에 프로그래밍 교육 과정에서 객체지향 프로그래밍을 교육하기 위한 프로그래밍 언어로 Java를 선택하고 있다. 먼저 이렇게 객체지향 프로그래밍 교육용 언어로 인기가 좋은 Java의 특징을 살펴봄으로써 교육용 언어로서의 Java의 이용을 고려하고자 한다.

Java는 인터넷 분산 환경에서 효과적으로 응용 프로그램을 작성할 수 있도록 설계된 객체지향 언어로 인터넷과 웹에서 보편적으로 사용되어 인터넷 기능과 환경에 많은 변화를 주었다[김형국외99]. 컴파일과 인터프리팅이 혼재하는 모델을 채택하고 있으며, Java를 공식적으로 발표한 선 마이크로 시스템사의 3)Java White Paper에 나타난 자바의 특징은 다음과 같다.

3) James Gosling과 Henry McGilton에 의해 쓰여진 것으로 자바 프로그래밍 언어의 설계 목표와 장점, 특성에 대해 상세하고 설명하고 있는 논문임

첫째, 간단하고 객체지향적이고 친숙하다. 자바는 C와 C++에서 많은 특성을 가져 왔기 때문에 C와 C++ 프로그래머에게도 친숙한 언어가 될 수 있으면서 포인터 연산을 사용하지 않으며, 연산자 오버로딩, 다중 상속 등의 기능을 삭제하여 C++의 복잡성을 제거하여 구문 자체가 단순하다. 또 사용자 인터페이스, 그래픽, 네트워크 등을 지원하는 풍부한 클래스 라이브러리를 지원함으로써 사용자가 직접 프로그래밍해야 하는 번거로움을 많이 줄여준다.

둘째, 견고성과 보안성이 높다. 포인터 처리를 없앴으로써 런타임 에러 발생을 감소시켰다. 자바 코드는 바이러스, 파일의 삭제나 수정, 데이터 파괴작업이나 컴퓨터 오류 연산 등을 방지할 수 있는 환경에서 실행되도록 설계되었다. 네트워크를 통해 PC로 다운로드된 자바 프로그램은 철저한 바이트 코드 검증 단계를 거치면서 보안에 어긋나는 코드가 숨어 있으면 에러를 발생시킨다.

셋째, 아키텍처에 독립적이고 이식성이 높다. 네트워크를 통해 프로그램을 다운 받아 하드웨어에 관계없이 사용하려면 근본적으로 아키텍처에 독립적이고 이식성을 보장하는 구조가 요구된다. 자바 프로그램은 하드웨어 아키텍처, 운영체제 인터페이스, 윈도우 시스템에 독립적인 바이트 코드 형태로 컴파일 되고, 자바 가상 기계에 의해 어떤 종류의 시스템에서도 해석되어 실행된다.

이와 같은 특징을 가지고 있는 Java를 객체지향 프로그래밍 교수·학습을 위한 교육용 언어의 관점에서 검토해 보면 긍정적인 측면과 부정적인 측면을 모두 가지고 있는데 먼저 긍정적인 측면을 보면 다음과 같다.

첫째, Java는 안전한 레퍼런스를 사용하여 쓰레기 수집(garbage collection)에 의해 메모리가 자동으로 관리되기 때문에 학습자가 메모리를 직접 접근할 코드를 직접 쓸 필요가 없어 학습의 부담을 덜어 준다.

둘째, 최근 인터넷 시대에 적합한 객체지향성, 포인터 연산 불허, 쓰레기 수집, 보안, 인터넷 기반 프로그래밍과 같은 실용성을 모두 갖추고 있어 객체지향 프로그래밍 학습이 현실 사회가 요구하는 실용적인 언어로 이루어질 수 있다.

이에 비해 Java는 객체지향 교육용 언어로서 부정적인 측면은 첫째 Java는 배우기 쉬운 언어가 아니기 때문에 객체지향 프로그래밍 교육이 객체지향 프로그램을 어떻게 설계하고 구성하는지 배우는 것임에도 불구하고 Java의 기본 구문을 배우기 위해 많은 시간이 소비될 수 있고, 둘째 교육용 언어로 Java를 이용하여 가르쳐 온 경험이 적기 때문에 Java를 학습하는 도구나 교수·학습 자료들이 충분하지 않고, 셋째 Java는 현재 놀라운 속도로 성장하고 개발되는 과정에 있기 때문에 교육용 언어로는 불안정한 면이 있다[Hong98].

본 연구의 목적이 객체지향 프로그래밍에 대한 기초 능력 배양을 위하여 객체지향 프로그래밍을 교수·학습하는데 있는 것이기 때문에 이에 적합한 객체지향 프로그래밍 언어를 선택하는 것은 중요하다. 프로그래밍 교육을 위해 프로그래밍 기본 개념에 충실한 Logo와 같은 특정한 교육용 프로그래밍 언어를 이용할 수도 있으나 이것은 작고 단순한 프로그래밍을 하도록 할 가능성이 크기 때문에 학문적 기초뿐만 아니라, 사회에서 널리 사용되고 있는 실용적인 특성을 갖춘 언어로 객체지향 프로그래밍을 다루는 것도의의가 있다고 본다. 이에 본 연구에서는 이와 같은 Java의 특징과 긍정적·부정적 측면을 모두 고려하여 학문성과 실용성 두 가지 측면을 모두 갖추고 있는 Java를 객체지향 프로그래밍 기초 능력을 기르기 위한 교육을 위한 프로그래밍 언어로 다루고자 한다.

3. 객체지향 교육의 유의점

객체지향 프로그래밍 교육에 있어서 교육용 언어로서의 여러 특징을 가지고 있는 Java를 가르칠 때 몇 가지 유의할 사항에 대한 주장들을 살펴보면 다음과 같다.

첫째, 기본형(primitive data)을 만드는 것은 객체에 대한 레퍼런스를 만드는 것이 아니라는 것을 다루어야 한다. 프리미티브 데이터 타입과 객체 즉 레퍼런스 타입이 둘 다 별개로 사용되는 것이다. 객체와 프리미티브 타입을 구별하는 것은 자바의 특징으로 객체지향 프로그래밍의 특징이 아니다.

둘째, 전통적인 예제 프로그램인 'Hello World' 프로그램은 객체지향 프로그램의 특징인 객체를 생성하는 것이라기보다는 메소드를 실행하는 것으로 이루어져 있다. 이 프로그램의 System.out.println문은 하나의 객체(System.out)로만 구성되어 있고 객체가 제공하는 서비스들 중에서 하나(println)를 이용하는 것을 보여주고 있다[Clark&MacNish&Royle98].

이와 같은 내용에 추가하여 객체지향에 대해 잘못된 시각을 지적하고 객체지향 프로그래밍 교수·학습에서 바르게 다루어져야 할 내용을 살펴보면 다음 몇 가지로 정리해 볼 수 있다.

첫째, 객체지향 개념은 상대적으로 새로운 것이라는 오류가 있다. 최초의 객체지향 언어 Simula는 1962년과 1967년 사이에 개발되었고, Smalltalk는 25년 전에 개발된 언어로 현재 우리가 사용하고 있는 언어보다 더욱 객체지향 언어이다. 이에 비해서 최근 교육용 언어로 인기 있는 Java가 1995년에 발표되어 상대적으로 새로운 객체지향 언어이지만 객체지향 개념 전체가 새로운 것으로 생각하는 것은 비약된 것이다.

둘째, 객체지향 개념과 절차지향 개념은 상호 배타적이라고 잘못 생각하는 경향이 있다. 절차적 추상화가 객체지향 접근에서 제거되는 것이 아니라 단지 객체 추상화 내부에 스며들어 있는 것이다. 하나의 객체 내에서 효과적인 내부 객체 설계를 만들기 위해 메소드는 필요한 만큼 분해되어 절차적인 것이 되어야 한다.

셋째, 제어구문과 객체는 서로 상충하는 개념이라는 오류가 있다. 일부의 객체지향 교육자들은 클래스와 메소드를 프로그래밍 하는 것을 제어구조보다 먼저 학습해야 한다고 주장하고 또 다른 교육자들은 제어구조들 없이 정의된 객체는 흥미 없는 예들이 된다고 주장한다. 조건문과 루프와 같은 제어구문과 제어구조들 자체가 절차적 관념을 강화하는 것은 아니며 단지 기능을 정의하는 저 수준의 구문일 뿐이다. 따라서 제어구조에 집중된 교육이 객체지향 접근과 서로 상충된다는 주장은 부적절하다.

넷째, C++는 객체지향 언어이고 Java는 순수한 객체지향 언어라는 오류가 있다. 그러나 C++는 순수한 절차지향과 순수한 객체지향을 혼합한 하이브리드 언어이다. 진정한 객체지향을 목표로 가정한다면 C++를 교육용 언어로서 이용할 경우 교육자는 그 언어에 있지 않는 객체지향 개념들을 찾아내어 가르쳐 주어야 할 책임이 있다. 위에서도 지적하였듯이 Java는 프리미티브 타입이 존재하므로 완전히 순수하지는 않고 객체지향에 가까운 언어라고 볼 수 있다[Lewis00, Holand&riffiths&Woodman97].

4. 객체지향 프로그래밍 교수·학습의 내용

개념에 대한 고전적인 생각은 이름, 내연, 외연을 다룬다. 이름은 개념을 표시하는 것이고, 내연은 개념에 의해 커버되는 현상을 성격 지우는 속성이며, 외연은 개념에 의해 커버되는 현상들을 말한다. 개념은 현상의 차이

점은 버리고 비슷한 속성에 초점을 맞추는 추상화에 의해 만들어지며, 추상화에는 3가지 하위기능이 있다. 이들 중 가장 기본적인 것이 분류화(classification)이며, 이 분류화는 개념에 의해 커버되는 현상을 정의하기 위해 사용된다. 분류의 역으로 하위 기능은 예시화(exemplification)이다. 개념은 자주 다른 개념들에 의해 정의된다. 이 하위 기능을 집합(aggregation)이라 하고 그 역의 하위 기능은 분해(decomposition)라고 부른다. 개념은 분류의 계층구조로 조직될 수 있다. 어떤 개념은 개념집합의 일반화로 여겨질 수 있다.

객체지향 프로그래밍에서 프로그램의 실행은 실세계의 실제나 상상적 부분의 행동을 시뮬레이션하는 물리적 모델로서 여겨진다. 물리적 모델은 현상과 개념이라는 관점에서 현실성의 개념에 기초한다.

정보처리과정에서 현상의 영역은 실체, 실체를 측정할 수 있는 속성, 실체에 대한 변형으로 확인된다. 실체는 시간과 공간에서 나타나는 양, 위치로 규정지어지고, 실체는 측정할 수 있는 특성을 가지고, 이 측정은 비교될 수 있고 타입과 값으로 설명된다. 실체에 대한 변형은 측정할 수 있는 속성을 변화시키는 사건들이 순차적으로 된 것이다. 실체와 측정 가능한 특성들과 변형은 정보처리과정에서 현상에 관련된 측면이다. 모델링되고 있는 현상에 대한 본질적인 속성들을 포착하기 위해서 추상화나 개념을 개발하는 것이 필수적이다.

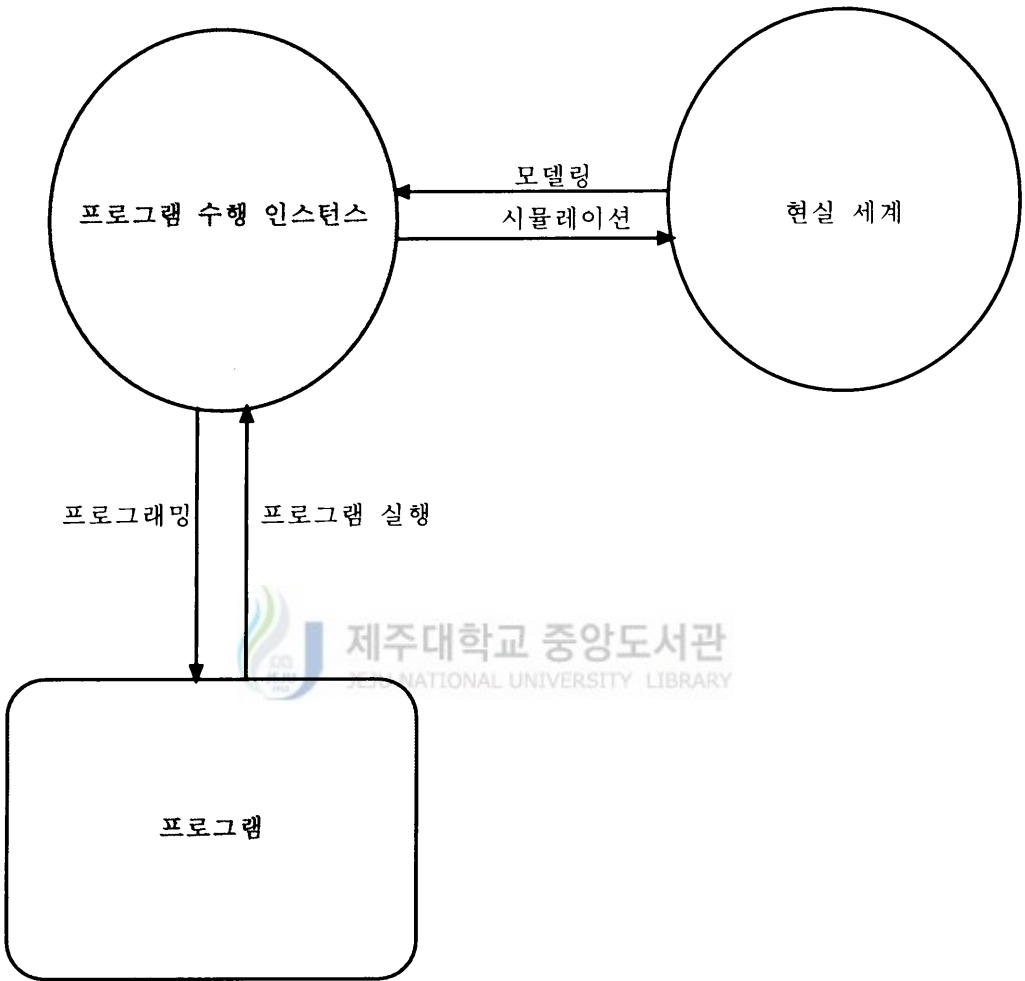
물리적 모델의 요소는 속성과 활동을 가진 객체들로 모델링된다. 실세계의 객체지향은 물리적 모델이며, 관련된 현상과 이 현상들이 가지고 있는 속성들을 선택함으로써 만들어진다. 물리적 모델은 객체들로 구성되며, 각 객체는 속성과 순차적으로 된 행동으로 특성화된다. 객체들은 현상의 물질 측면을 조직하고 물질에 대한 변형들은 행동을 실행시키는 객체들에 의해 반영된다. 객체는 부분 객체들을 가질 수 있고, 속성은 부분 객체나 별개의

객체에 대한 레퍼런스일 수 있다. 어떤 속성들은 객체의 측정할 수 있는 특성들을 포함한다. 주어진 시간에서 객체의 상태는 그 실체와 측정할 수 있는 속성과 그 다음에 계속되는 행동에 의해 표현되고, 전체적 모델의 상태는 이 객체들의 상태를 말한다.

분류화는 대부분의 프로그래밍 언어들과 타입과 절차와 클래스에 의해 지원 받는다. 집합/분해도 대부분의 프로그래밍 언어에 의해 지원 받는다. 하나의 절차는 다른 절차에 의해 정의될 수 있고 타입은 다른 타입에 관련하여 정의될 수 있다.

일반화(generalization)/상세화(specialization)(서브클래싱 또는 상속이라 부름)를 지원하는 언어 구조는 객체지향(object-orientation)을 지원하는 프로그래밍 언어의 주요한 특성으로서 언급된다. Simula에서 상속이 소개되었으나 최근에는 상속이 객체지향 프로그래밍에 주요하게 결합되었다. 일반화는 클래스와 하나 이상의 서브 클래스와의 관계이고 서브 클래스는 부모 클래스 보다 더욱 구체화된다. 상속, 일반화, 구체화는 서로 관련된 개념이다[Madsen&Møller-Perderson88].

현실 세계의 현상은 동적으로 상호 작용하는 객체로 이루어지며 이러한 현상을 컴퓨터에서 그대로 흉내내기 위하여 객체를 발견하고 추상화하는 것은 현실 세계를 모델링 하는 것이다. 이러한 모델링을 정적으로 구현한 것이 프로그램 코드이고 이것이 컴퓨터 내부에서 동적으로 실행되어 사용자에게 실행 결과를 돌려준다. 객체지향 프로그래밍을 학습하는 것은 현실 세계의 현상을 모델링한 것을 정적인 코드로 구현하는 방법과 이 정적인 코드가 컴퓨터에서 실행되는 동적인 상황과의 연계성을 배우는 것이라 볼 수 있다. 즉 객체지향 개념이 프로그램 관련 수행 영역과 구현 영역에 어떻게 다른지 인식하고 이들 두 영역 간 연계를 이해하는 것이라고 볼 수 있다. 이와 같은 객체지향 프로그래밍은 [그림 1]과 같이 제시될 수 있다.



[그림 1] 객체지향 프로그래밍

객체지향 프로그래밍 교수·학습 내용은 객체와 클래스의 개념과 실제로 프로그래밍 하는 것으로 구분되며 프로그램의 정적 구조와 동적 구조 사이의 미묘한 차이를 구별하는 것을 포함한다. 정적인 프로그램 구조는 소스 코드에 의해 주어지는 클래스의 정의라고 볼 수 있고, 수행 시간 구조는

실행하는 동안 생성되는 객체들과 그들의 상호작용으로 이루어진다. 그러므로 학습자는 정적인 구조를 프로그래밍 하면서도 실제적으로는 프로그램 실행 시간 구조를 심적(mental)으로 시각화하는 작업을 동시에 해야 한다[Clark&MacNish&Royle98].

이와 같은 객체지향 프로그래밍 학습의 성격에 따라 본 연구는 객체지향 프로그래밍 관련 내용을 크게 객체와 클래스, 객체 간 상호작용, 클래스 간 관계로 구분하고, 객체와 클래스에서는 객체의 구성과 클래스의 구성, 객체와 클래스와의 관계를 살펴보고, 객체 간 상호작용에서는 객체 간 참조관계와 호출관계를 살펴보고, 클래스 간 관계에서는 상속과 다형성에 대해서 살펴보았다.

4.1. 객체와 클래스



4.1.1. 객체

객체(Object)란 인간이 하나의 개념으로 파악하는 모든 것으로 개념적인 단위를 의미하며 관심이 있는 특정한 대상을 말한다. 객체는 뚜렷한 (crisp)경계를 가진 개념, 추상화, 사물(thing)이고 문제에 관한 의미이다[우치수98]. 이 객체는 유일한 이름을 가지고 있는 유일한 존재로 물리적 객체와 개념적 객체로 나뉜다. 실세계를 구성하는 요소들의 특성과 행동양식을 모델로 해서 만들어진 개념을 말하는 것이므로 결국 우리 주위에 있는 모든 것은 객체가 된다.

■ 객체의 생성

객체는 문제해결에 필요한 상태나 속성에 해당하는 자료구조와 이 자료구조를 다룰 수 있는 함수들로 표현되는 소프트웨어 모듈로서 임의의 한 클래스로부터 그 클래스의 자료구조와 연산을 가지고 생성된다. 이것을 클

래스의 인스턴스화라고 부르며, 클래스가 구현된 클래스의 활성체로서 실제로 컴퓨터 내에서 수행된다.

■ 객체의 사용

객체는 정보처리의 주체로서 메시지 전달 혹은 메소드 호출을 통해 상호작용 한다. 객체는 원자적(atomic)단위로서 객체의 사용자들에 의해 분리될 수 없고 객체 구현에 대한 직접 접근은 차단되어 못하게 되고 그 객체가 지닌 메소드들에 의해 간접적으로만 접근할 수 있는데, 이것이 정보은닉이다. 즉 프로그래머가 객체를 사용하기 위하여 메소드가 객체 내에서 정확히 어떤 과정으로 수행되는지 알 필요가 없으며 메소드의 기능만 정확히 알고 있으면 메소드를 호출하여 객체를 이용할 수 있다. 그 객체의 변수들을 읽거나 쓰기 위해 그 객체의 메소드를 호출하여 객체 자체 내에서 문제를 해결하도록 하는 것이다. 객체에게 메시지를 보내면(메소드를 호출하면) 클래스에 정의된 메소드가 인스턴스를 대상으로 수행되며 이를 통해 인스턴스 내의 상태를 설정·변경할 수 있다. 즉 인스턴스의 메소드를 이용해서 메모리 값을 설정·변경할 수 있다. 객체는 서로에게 파라미터로써 전달될 수도 있다.

■ 객체의 소멸

객체는 소멸되는 존재이다. 소멸시점은 일반적으로 사용자가 명시적으로 객체 제거를 요구한 때이거나, 객체에 대한 레퍼런스가 모두 사라져 필요 없는 객체라 판단할 수 있는 시점이다. 개념적으로 객체는 메모리 공간에서 사라져 더 이상 그 객체에 접근할 수 없게 된다.

4.1.2. 클래스

클래스는 공통된 성향이나 취향에 의해 서로 그룹지어지는 사람이나 사물이다. 현실세계에서 사물은 어떤 유형을 가지고 있으므로 이 사물들은

같은 유형을 가지고 있는 사물들로 구분 지을 수 있다. 객체지향에서는 각 사물이 어떤 유형에 속하는가 하는 것은 클래스를 통하여 표현할 수 있다. 이때 클래스는 같은 형태의 객체들의 집합으로 그 객체들을 이루고 있는 데이터 구조와 객체들이 공유하는 메소드를 포함하며 객체의 상태(변수), 행동(메소드)를 정의하고 있는 설계도, 즉 프로토타입으로서 데이터형의 선언과 구현을 동시에 제공한다. 클래스의 데이터를 필드(field)라고 하고 필드에 작용하는 연산을 메소드라고 한다.

이에 비해 추상자료형은 구현과 세부 사항을 명시하지 않고 자료 추상화와 인터페이스를 정의한다. 추상자료형은 여러 가지 목적에 맞출 수 있고 다양한 환경에 적합할 수 있다. 객체지향 프로그래밍에서는 추상화를 위해 클래스라는 개념을 사용한다. 클래스는 이 추상 자료형의 구현부가 된다. 클래스를 통해 구체적인 원시 데이터가 아닌 추상화된 데이터를 다루게 되고 복잡한 데이터를 보다 효과적으로 다룰 수 있다. 클래스는 구현과 관련된 세부 사항을 명시하지 않고 자료추상화와 인터페이스를 정의하는 추상자료형(ADT)에 상속과 다형성의 개념을 추가한 것이다. 클래스가 정의되면 객체 선언의 타입(type)으로 이용되며 클래스 이름이 새로운 타입의 이름이 된다. 클래스들은 하나의 계층구조 내에서 정렬되며 클래스는 새로운 객체를 만들기 위해 정의된 것으로 프로그램 연산에 거의 관련되지 않는다. 클래스의 계층구조는 새로운 클래스를 만드는 것을 편리하게 하면서도 객체에 관해서는 거의 무관하다[Armstrong99].

4.1.3. 클래스와 객체의 관계

■ 클래스와 객체의 차이

객체지향 프로그래밍에서 객체를 만들기 위해서는 반드시 클래스라는 것을 통하도록 되어있다. 클래스는 객체의 형태와 템플릿(형틀)이다. 클래스

는 정적인 것이나 그 클래스의 특성을 외부로 노출하여 동적인 행위를 하는 것이 객체이다. 클래스는 객체를 통해서만 실현되고, 객체는 클래스를 통해서만 바르게 이해된다. 특정 클래스를 실제 사용할 수 있도록 변수 선언한 것이 그 클래스에 대한 인스턴스이고 이것이 실제 프로그램에 사용 가능한 객체이다. 클래스 기반 언어에서 객체를 바로 표현하는 방법은 없으며 반드시 클래스를 통해 객체가 생성된다. 클래스는 객체의 구조와 기능을 기술하기 위해 사용되는 개념이다. 클래스가 추상적, 종합적인 것이라면 객체는 구체적이고 분석적인 것이다. 클래스는 객체를 만드는 모체인데 반해 객체는 어떤 클래스로부터 만들어진 인스턴스이다. 클래스는 객체들의 공통점을 모아 두었다면 객체는 클래스가 가지고 있는 공통점에다 자신만의 고유한 상태로 특성화시킨 것이다.

■ 인스턴스 변수와 클래스 변수

인스턴스 변수나 인스턴스 메소드는 클래스의 각각의 객체에 속하는 것이다. 클래스 변수나 클래스 메소드는 클래스 그 자체에만 속한 것이다. 클래스 변수는 객체의 생성과 관련이 없으며, 객체를 생성하지 않더라도 클래스 변수에 대한 접근이 가능하고, 클래스 메소드는 객체에 속한 것이 아니므로 객체에 대한 참조값이 필요 없다. 인스턴스 메소드는 클래스에 속한 모든 인스턴스 객체들에게 공통으로 적용된다.

클래스 변수와 클래스 메소드는 모두 유일하게 정해진 것이다. 인스턴스 변수는 각각의 객체들이 그 자신의 변수들을 가지고 있어야 할 때 이용한다. 인스턴스 변수와 인스턴스 메소드는 독립하여 존재할 수 없고 인스턴스의 일부만 존재한다.

4.2. 객체 간 상호작용

객체 간의 상호작용은 특정 상황과 연관시켜 이해해야 한다. 이는 상황별로 그와 관련된 객체들이 서로 협력하여 각 객체 나름의 행동을 통해 반응하게 되고, 그 과정에서 객체들의 상태 변화가 수반되기 때문이다. 따라서 객체 간 상호작용을 보다 체계적으로 이해하려면 다음 세 가지 요소를 이해해야 한다. 특정 상황과 관련된 객체들의 범주, 관련 객체들이 서로를 대상으로 행하는 작업의 내용, 이들 작업의 결과로 각 객체가 보이는 행동 및 상태 변화가 이해해야 할 요소들이다.

4.2.1. 객체 간 관계

프로그램 수행 과정에서 생성된 객체들은 소멸되기까지 다른 객체들과의 관계를 통해 자신에게 맡겨진 역할을 수행한다. 객체 간의 관계는 참조관계와 호출관계로 구분하여 이해할 수 있다. 참조관계는 객체들이 서로 간의 레퍼런스(*reference*)를 인식(한 객체가 인스턴스 변수로서 다른 객체에 대한 레퍼런스를 지님)하고 있을 때 설정되는 관계이다. 객체를 정점(*vertex*)으로 나타내고 참조관계를 유향 에지(*directed edge*)로 표현한다고 할 때, 프로그램 수행 중 한 시점의 스냅 사진은 그 당시 존재하는 모든 객체 간의 참조관계를 유향그래프(*directed graph*)로 보여 주게 된다. 이에 반해 호출관계는 한 객체가 다른 객체에 대한 레퍼런스를 이용하여 그 메소드를 호출할 때 설정되는 일시적 관계이다. 객체는 자신의 인스턴스 변수에 저장된 레퍼런스를 이용하거나 다른 객체가 지닌 레퍼런스를 전달받아 대상 객체를 호출하게 되므로, 객체 간 참조관계를 객체 간 호출관계의 근간 혹은 잠재적 호출관계로 이해할 수도 있다.

일반적으로 참조관계가 호출관계보다 정적(*static*)이므로 이를 통해 특정

상황과 연관된 객체의 범주를 효과적으로 파악할 수 있으며, 참조 관계에 바탕한 호출관계를 이해할 때 관련 객체들이 서로를 대상으로 행하는 작업의 내용을 파악할 수 있다. 객체는 생성될 때부터 다른 객체와 참조관계를 맺는데, 모든 참조관계가 제거되는 상황이 바로 객체의 소멸을 의미한다. 따라서 객체가 생성될 때부터 소멸될 때까지 맺게 되는 참조관계가 어떻게 변화되어 가는지와, 참조관계를 기반으로 다른 객체와 어떤 호출관계를 맺어 가는지를 이해하는 것은, 비가시적인 프로그램 수행 세계를 학습하는데 있어 매우 중요한 요소라 하겠다.

4.2.2. 객체의 상태 변화

객체는 특정 시점에서 상태(인스턴스 변수들의 값)를 갖는다. 객체의 상태는 그 클래스에 정의된 메소드를 수행할 때 해당 객체가 나타내는 행동양식(*behavior*)의 기반이 된다. 객체가 그 당시 어떤 상태인지 알지 못하면 메소드 수행 결과를 정확히 이해하고 예측할 수 없게 된다. 객체 간 상호작용은 궁극적으로 특정 객체에 대한 호출과 해당 객체의 행동으로 나타난다. 따라서 특정 시점에서의 객체 상태가 객체 행동양식에 어떤 영향을 주는지와, 특정 메소드 수행에 의해 객체가 어떤 상태변화를 겪는지에 대한 학습은 객체 간 상호작용을 이해하기 위해 필요한 기본 요소라 할 수 있다.

4.3. 클래스 간 관계

4.3.1. 상속

실세계에서 상속이라는 단어는 이전 세대에서 획득한 것을 의미한다. 프로그래밍 세계에서 프로그래머들은 기존 코드와 매우 비슷하게 코드를 작

성할 수 있는데 이것은 주로 코드를 자르고 붙임으로써 이루어졌고, 이 유사성을 포착하여 형식화할 필요성에 의해 생긴 것이 상속이다. 객체지향 프로그래밍에서 새로운 클래스는 하위 클래스로 이것은 상위 클래스를 상속한다고 하며 모든 클래스는 반드시 어떤 클래스에서 파생되어야 한다. 상속은 객체의 기능을 확장시킬 수 있도록 하여 원래 타입의 속성들로부터 확장된 새로운 타입을 만들 수 있다. 상속은 객체지향 프로그래밍의 중요한 원리로 기존의 코드를 재사용하고 쉽게 확장함으로써 코드 작성에 드는 시간과 노력을 줄이는 데 있다.

객체지향 프로그래밍에서 클래스들은 하나의 클래스 계층구조로 조직되어, 한 클래스와 관련된 다른 클래스에서 그 클래스의 코드를 재사용할 수 있게 하는 형태로 나타난다. 이렇게 계층구조를 만들기 위해 상속을 이용하는 것은 객체지향 프로그램을 그 보다 좀 더 단순한 객체기반 프로그래밍과 차별화 시켜주는 부분이다.

상속의 문제점은 첫째로 상부 클래스 구현의 변경이 모든 하부 클래스 구현의 변경에 직접적인 영향을 줄 수 있다는 것이다. 슈퍼 클래스 구현의 개선이 서브 클래스들에게 자동적으로 전파되는 효과도 있지만, 슈퍼 클래스의 구현을 직접적으로 상속받아 사용하는 서브 클래스들을 집단적으로 파괴하는 경우도 있다. 두 번째 문제로 모듈간의 결합도를 가능한 낮추는 것이 좋은 프로그램이 되기 위한 조건인데 상속은 클래스 간의 본질적인 결합을 의미한다. 또한 메소드를 실행할 때 현재 클래스의 메소드를 조사하여, 지역변수와 매개변수를 찾고 그 다음 슈퍼 클래스를 검토하며, 더 이상 검토할 슈퍼 클래스가 없을 때까지 클래스 계층을 계속 검토한다. 이와 같이 상속은 어떤 메시지에 대하여 클래스 계층을 오르내리며 메소드를 검색해야 하는 경우가 자주 발생하는 문제점을 가지고 있기 때문에 어떤 클래스의 코드를 이해하려면 그 클래스의 정적인 의미뿐만 아니라 실행 시에

발생할 수 있는 동적 바인딩 효과까지 이해해야 한다. 즉 하부 클래스를 읽을 때 상속된 멤버가 보이지 않지만 존재하므로 혼동하지 않도록 해야한다.

4.3.2. 다형성

실세계에서 다형성은 “여러 형태를 가질 수 있다” 또는 “여러 형태로 구현될 수 있다”는 의미이다. 객체지향에서의 다형성(polymorphism)은 공통의 인터페이스를 사용하여 여러 개의 기능을 수행할 수 있는 것으로 동일한 이름의 함수가 객체형을 바탕으로 서로 다른 행동양식을 갖도록 하는 과정이다. 다형성은 상속의 일부분으로서 구현되어 소스 코드의 재사용을 강력하게 지원한다. 포괄적인 형태로 프로그램 코드를 작성함으로써 같은 모양의 코드가 다른 여러 의미를 가질 수 있도록 하고 시스템의 새로운 기능을 추가하기 쉽게 함으로써 시스템의 확장 가능성을 높여 준다. 다형성은 공통되는 로직을 공유하는 클래스들을 구현할 수 있도록 한다. 주어진 클래스의 한 객체는 서브 클래스의 모든 형태에 적용할 수 있으며 서브 클래스는 그 슈퍼 클래스를 위해 대체될 수 있다.

다형성의 방법에는 재정의와 다중정의를 있으며 재정의는 상속 계층의 다른 클래스에서 인스턴스 메소드가 메소드의 이름, 메소드의 파라미터의 개수와 타입, 리턴 타입은 같은 것으로, 즉 상위 클래스와 하위 클래스에 같은 이름의 변수나 메소드가 있는 경우, 하위 클래스에서 상속할 수 없으며 하위 클래스의 메소드가 상위 클래스의 메소드를 덮어쓴다. 즉 하위 클래스가 상위 클래스의 인스턴스 메소드를 새롭게 구현할 수 있다.

다중정의를 하나의 클래스 안에서 이름이 같은 메소드가 여러 개의 다른 일을 하도록 하는 것이다. 이 때 메소드의 이름이 꼭 같더라도 메소드의 파라미터의 개수, 타입, 순서를 다르게 하여 일의 성격은 비슷하지만 여러 개의 다른 일을 하는 것을 나타낸다.

IV. 시각적 도구의 이용 방안

1. 도구의 특징

시각적 도구를 이용하기 위하여 분석 대상으로 삼고 있는 두 개의 객체지향 프로그래밍 시각적 도구인 BlueJ와 Java Demonstrations는 객체의 생성, 사용 과정을 시각화하고 있다.

BlueJ의 주요 특징은 객체지향 프로그래밍 시작을 애플리케이션 구조에 초점을 맞추도록 하여 프로그래밍 초보자들이 처음부터 코딩과 컴파일, 디버깅하는 과정에서 놓치기 쉬운 클래스 간 관계, 클래스와 객체와의 관계를 시각화하고 있다. 클래스를 4가지 유형으로 나누어 클래스 계층구조로 클래스 간 관계를 시각화함으로써 객체지향 프로그래밍의 특징인 상속을 시각화하고 있다. 또한 학습자가 인자 값을 입력하고 메소드를 호출하면 그 실행 결과를 보여 주는 절차를 통해 학습자와 BlueJ가 서로 상호 작용적으로 학습할 수 있다[Kölling&Rosenberg00].

이에 비해 Java Demonstrations는 객체의 메소드 실행 과정의 동적 영역을 시각화한 것으로 객체가 생성, 소멸, 사용되는 변화과정을 클래스 구현 코드와 객체 이미지를 연결시켜 애니메이션으로 제시하고 있다[Ellis00]. 이 도구는 BlueJ보다 객체의 생성과 상호작용에 대해 더 세부적 시각화하고 있어 복잡하게 보일 수도 있지만 프로그램이 실행되는 동적 과정을 이해하는 데 유용하다.

Java Demonstrations에서 클래스와 객체, 메소드의 실행과정과 클래스 구현 과정의 관계를 나타내는데 사용된 애니메이션의 효과를 살펴보면 애니메

이션은 학습과제가 실체와 현상의 시간에 따른 변화와 그들 움직임간의 인과관계 등에 초점이 맞추어져 있을 경우, 추상성을 낮추고, 주위의 불필요한 배경이나 부분들로부터 핵심부분들을 드러나게 해 준다[Meyer&Gallin90]. Mayer와 Anderson은 애니메이션이 개념이나 현상에 대한 구체적이고 직접적인 표현과 추상적 표현사이를 연결해 주는 다리 역할을 해주기 때문에, 그 개념에 좀 더 정확하고 분명한 심상을 만드는데 도움이 되어 전문가보다는 초보자들이 이용하는 것이 효과적이라고 보았다. 또한 같은 학습내용을 글과 그림으로 동시에 제공할 때 그림과 글 두 가지의 학습단서를 가질 수 있어 학습을 더욱 촉진시킬 수 있다고 주장하고 있다[Mayer&Anderson91]. 이와 같은 주장에 비추어 보았을 때, Java Demonstrations는 애니메이션과 함께 설명을 동시에 사용하고 있는 것으로 객체지향 프로그래밍에 대한 기본 개념들을 효과적으로 시각화하고 있다.

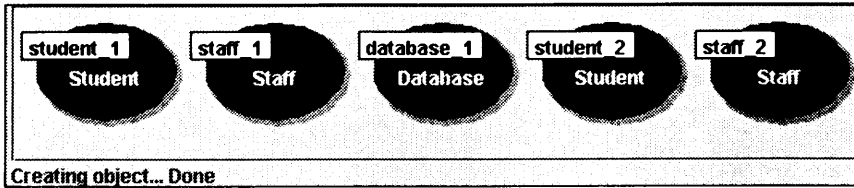
2. 시각화 방법의 비교 · 분석

객체지향 프로그래밍과 관련하여 객체와 클래스, 객체 간 상호 작용, 클래스 간의 관계가 시각적인 객체지향 프로그래밍 도구인 Java Demonstrations와 BlueJ에서 어떻게 시각화되고 있는지 항목별로 나누어 비교 · 분석함으로써 객체지향 프로그래밍 학습에 시각적인 도구를 이용하기 위한 방안을 마련하고자 한다.

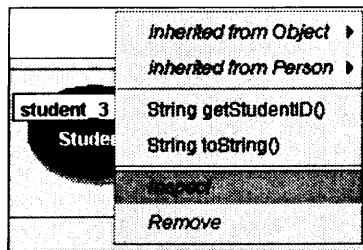
2.1. 객체와 클래스의 시각화

2.1.1. 객체 구성의 시각화

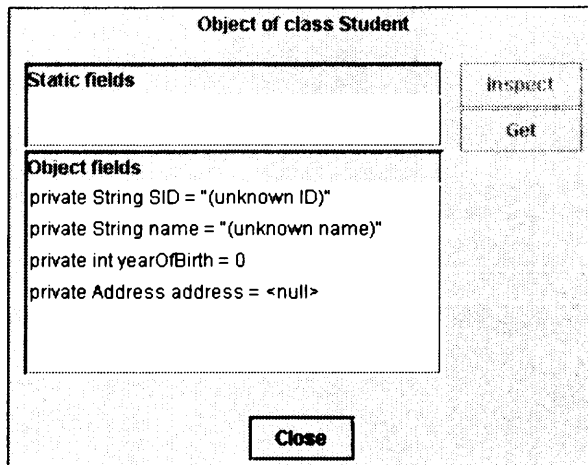
BlueJ에서 객체는 인스턴스 이름과 클래스 이름으로 구성되고[그림 2-a], 인스턴스 메소드[그림 2-b]와, 인스턴스 변수[그림 2-c]로 구체화된다.



[그림 2-a] 객체의 구성



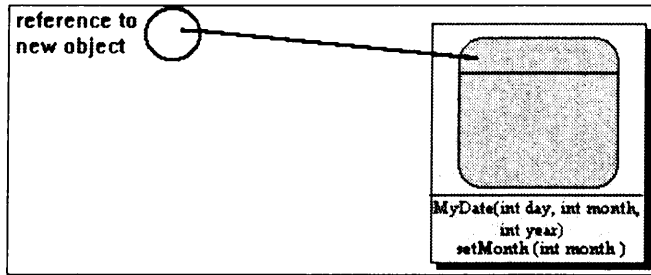
[그림 2-b] 인스턴스 메소드



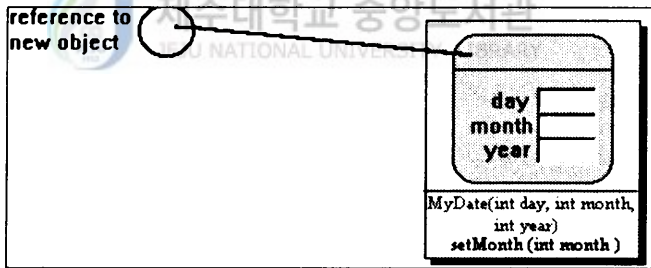
[그림 2-c] 객체의 인스턴스 변수

이에 비해 Java Demonstrations에서 객체는 [그림 3-a]와 같이 인스턴스 메소드, 인스턴스 변수, 레퍼런스로 구성된다. 객체 이미지에 [그림

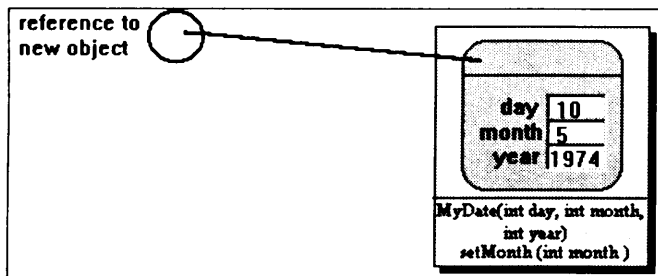
3-b)와 같이 인스턴스 변수의 이름과 [그림 3-c)에서처럼 값이 할당되는 객체 구성 과정이 시각화된다.



[그림 3-a] 객체 구성 초기화



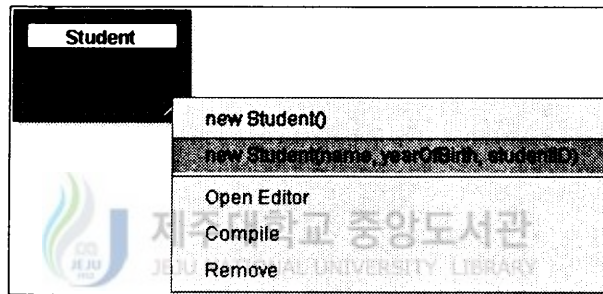
[그림 3-b] 객체 변수 이름 할당



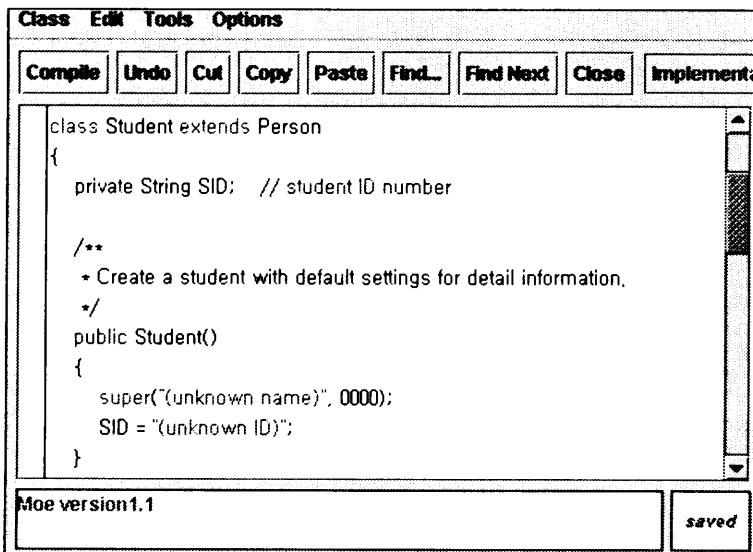
[그림 3-c] 객체 구성 완료

2.1.2. 클래스 구성의 시각화

BlueJ에서 [그림 4-a]와 같이 클래스는 이름과 클래스형에 따라 각각 다른 색의 사각형 아이콘이 되며, 생성자와 클래스 편집 창, 컴파일, 제거를 메뉴로서 구체화된다. 또한 [그림 4-b]와 같이 클래스의 편집 창에서, 클래스를 구현할 수 있는 키워드들이 다른 색을 가지게 되어 코드를 입력하고 컴파일하고 디버깅하는 것을 시각적으로 지원하고 있다.



[그림 4-a] 클래스 구성



[그림 4-b] 클래스 편집 창

이에 비해 Java Demonstrations에서 클래스는 [그림 5-a]와 같이 프로그램 실행 과정을 색깔 변화로 반영하는 정적인 코드로 구성되어 있고 코드의 실행과정과 관련 클래스는 버튼 클릭 결과 코드 창으로 팝업 된다. 메소드 코드가 점차 실행됨에 따라 메소드 호출 부분 코드는 [그림 5-b]와 같이 메소드의 실행 결과로 대체된다.

```

public class TestDateOfBirth {
    public static void main (String [] args)
    {
        generateDateOfBirth();
    }

    public static void generateDateOfBirth ()
    {
        MyDate birthday = new MyDate {10, 5, 1974};
        Person mary = new Person ("Mary Smith");
        mary.setDateOfBirth (birthday);
        mary.displayPerson ();
        birthday.setMonth (6);
        mary.displayPerson ();
    }
}

```

Show Class Person
Show Class MyDate

[그림 5-a] 코드의 실행과 관련 클래스

```

public class TestDateOfBirth {
    public static void main (String [] args)
    {
        generateDateOfBirth ();
    }

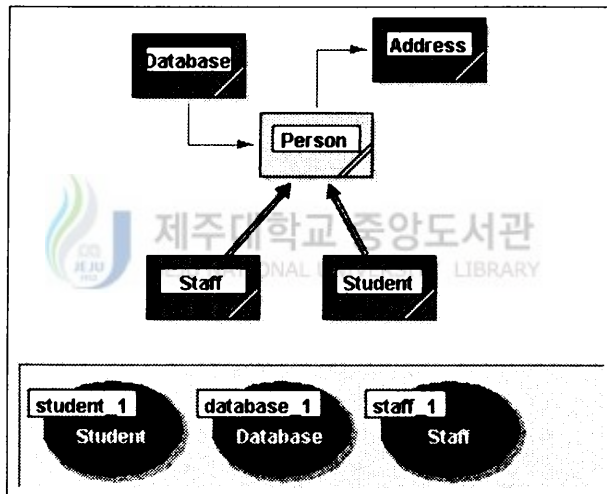
    public static void generateDateOfBirth ()
    {
        MyDate birthday = reference to new object ;
        Person mary = new Person ("Mary Smith");
        mary.setDateOfBirth (birthday);
        mary.displayPerson ();
        birthday.setMonth (6);
        mary.displayPerson ();
    }
}

```

[그림 5-b] 클래스 코드 대체

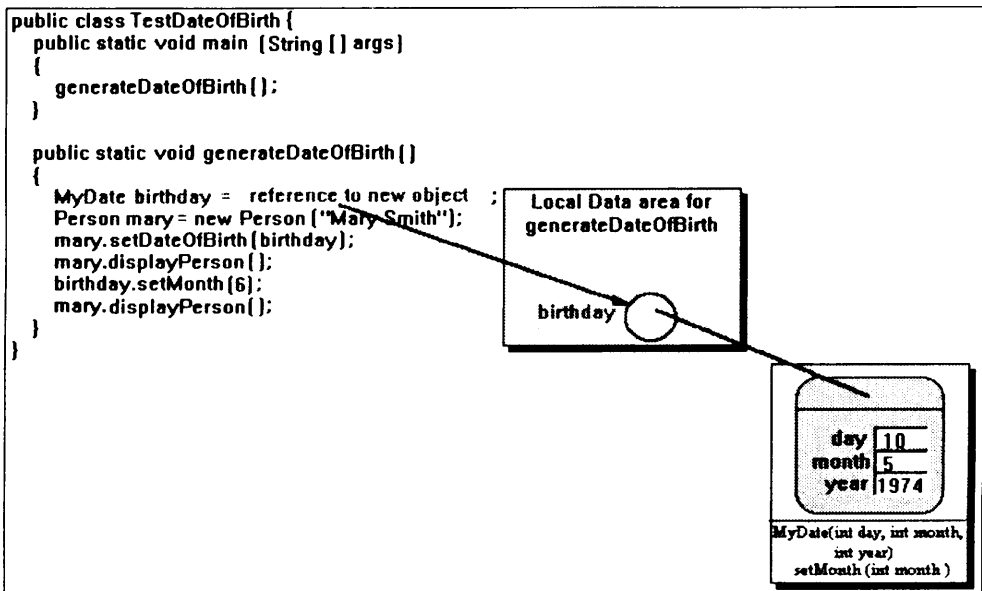
2.1.3. 클래스와 객체의 관계 시각화

BlueJ에서 클래스와 객체의 관계는 [그림 6-a]와 같이 메인 창에서 중앙의 클래스 아이콘과 하단의 객체 아이콘을 생성하는 것으로 시각화된다. 생성된 객체는 사용자가 소멸시키거나, 클래스 구현의 변화나, 클래스 재컴파일시에 객체 영역에서 사라짐으로써 그 객체를 다시 이용하고자 할 경우 객체를 재설정(reset)시키는 결과를 가져온다.



[그림 6-a] 클래스와 객체의 관계

이에 비해 Java Demonstrations에서는 [그림 7-a]와 같이 클래스와 객체의 관계는 클래스 구현 코드와 객체 이미지를 연결시키고 클래스 메소드의 동적 데이터 영역과 객체의 레퍼런스와의 연결 과정을 시각화하고 있다. 현재 실행되고 있는 클래스가 참조하고 있는 객체를 생성한 관련 클래스의 구현 창을 팝업 시킴으로써 클래스와 객체의 관계를 시각화시킨다.

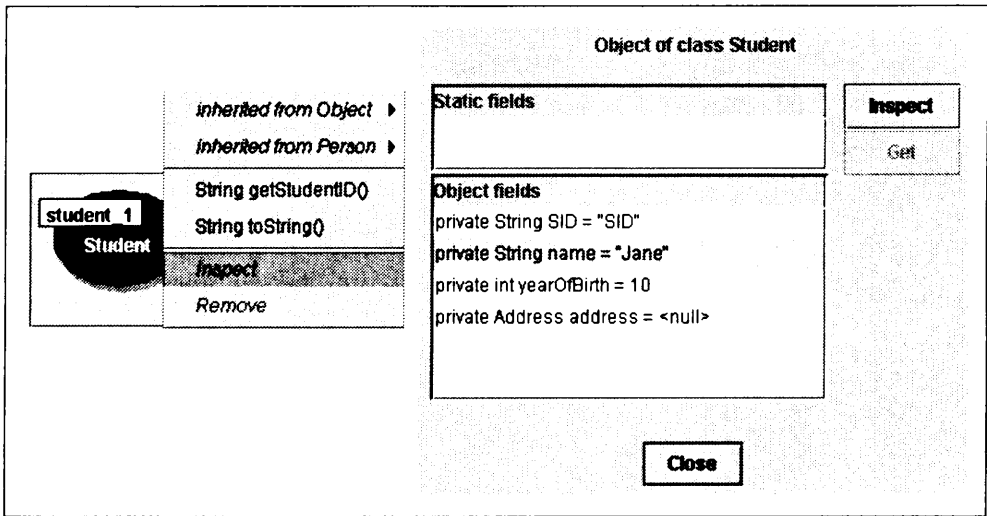


[그림 7-a] 클래스와 객체의 관계

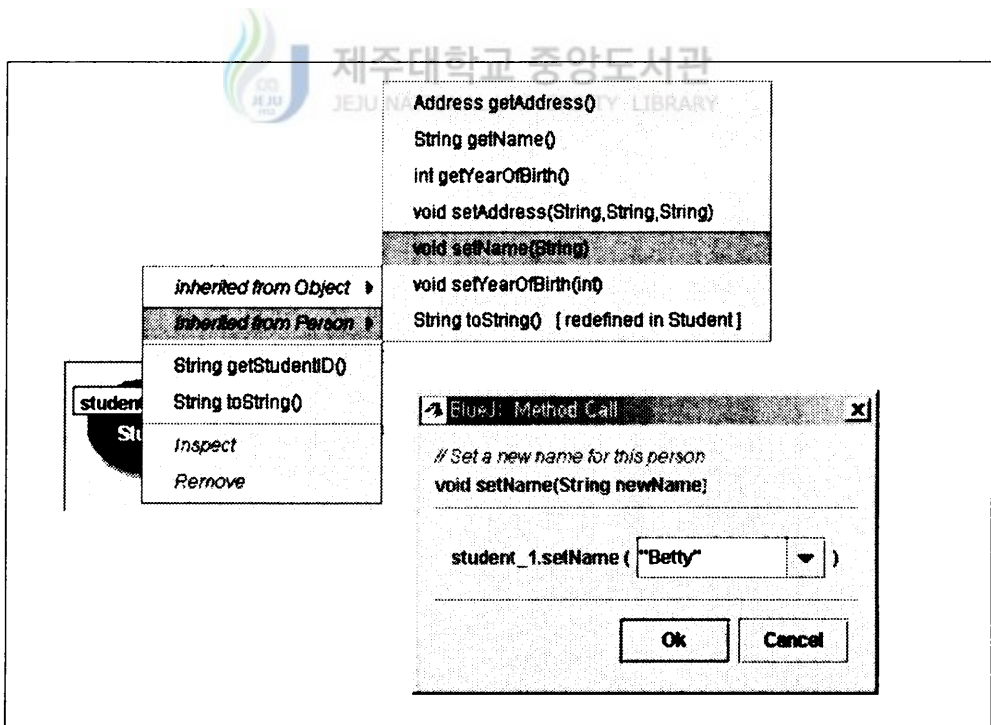


2.2. 객체 간 상호작용의 시각화

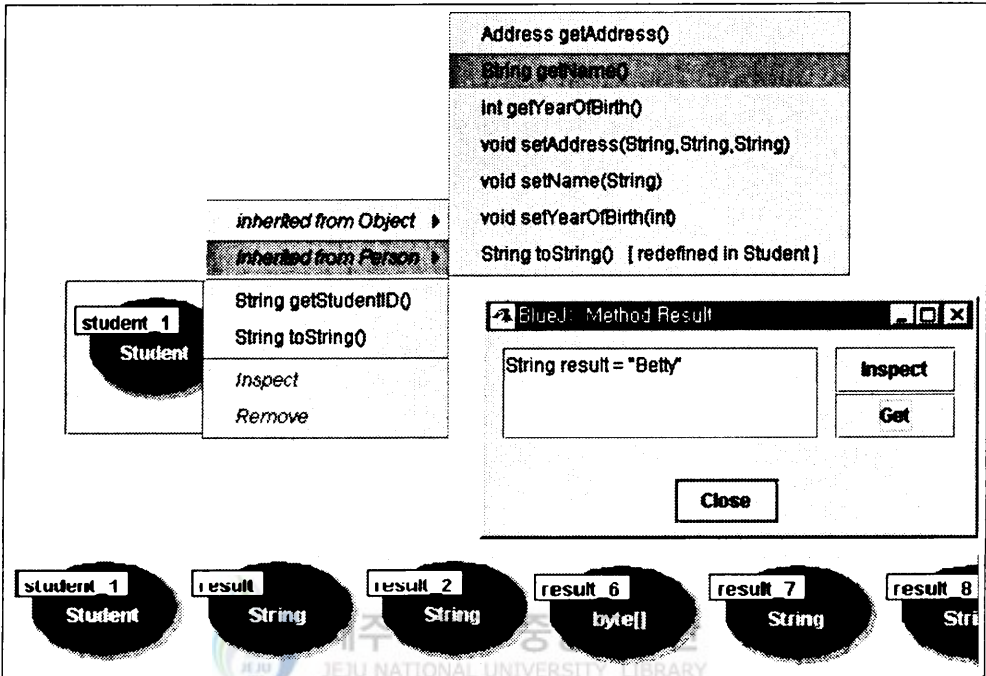
BlueJ에서는 객체의 상태변화는 다음 3단계 절차로 시각화된다. 변화시키고자 하는 객체의 상태 확인은 [그림 8-a]와 같은 방식으로 이루어지며 객체의 상태를 원하는 내용으로 변화시키는 과정은 [그림 8-b]와 같은 방식으로 시각화되고, 변화된 상태의 확인은 메소드 호출 결과로 [그림 8-c]의 방식으로 시각화한다. 그리고 객체의 메소드 결과가 특정한 프리미티브 값인 경우는 대화 상자 안에 1차적으로 시각화되고, 값이 아닌 별도의 출력 창 팝업[그림 8-d] 등 다양한 방식에 의해 시각화된다. 그리고 객체를 메소드 호출의 인자로 전달할 때는 해당 객체 아이콘에 마우스 클릭하면 된다. 즉 특정한 일을 하는 메소드를 가진 객체들을 찾아 클릭함으로써 그 객체를 참조하게 되지만 객체 간의 관계를 시각화하는 영역이 없어 이에 대한 연구가 필요하다.



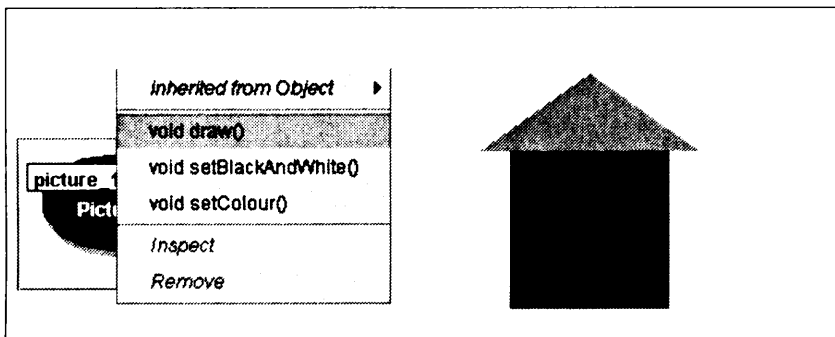
[그림 8-a] 객체의 상태 변화 A



[그림 8-b] 객체의 상태 변화 B



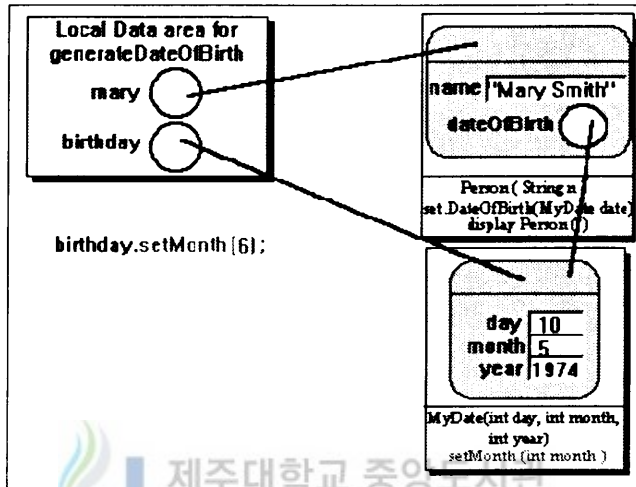
[그림 8-c] 객체의 상태 변화 C



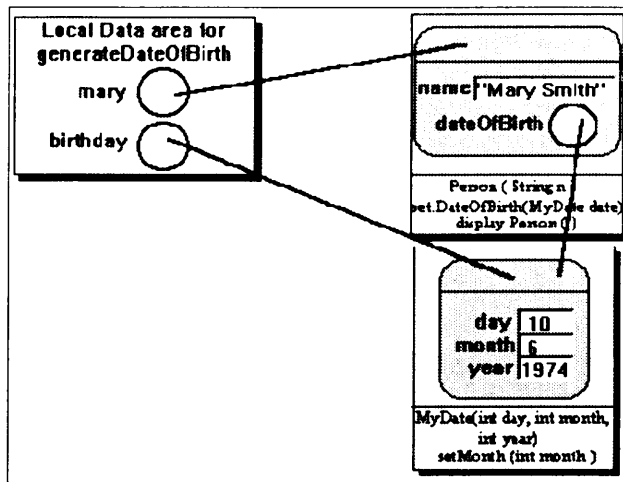
[그림 8-d] 객체의 메소드 호출 & 결과

Java Demonstrations에서 객체 간 참조관계와 객체의 상태 변화는 [그림 9-a], [그림 9-b]로 시각화되며, 객체 간 호출관계는 [그림 9-c]와 같이 시

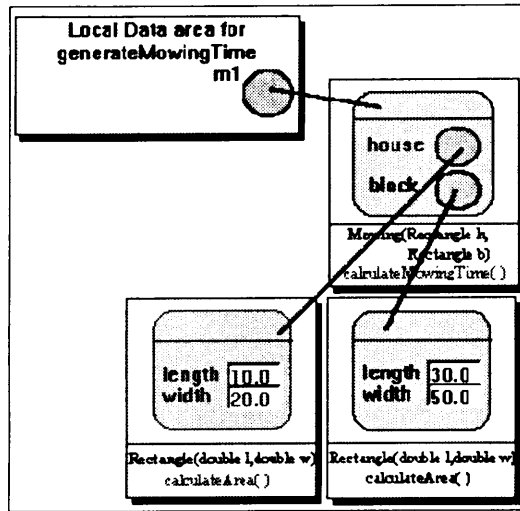
각화된다. 객체의 인스턴스 메소드가 호출되는 것은 객체 이미지의 메소드 시그니처의 색깔 변화를 통해 시각화된다.



[그림 9 a] 객체 간 참조관계와 상태 변화 A



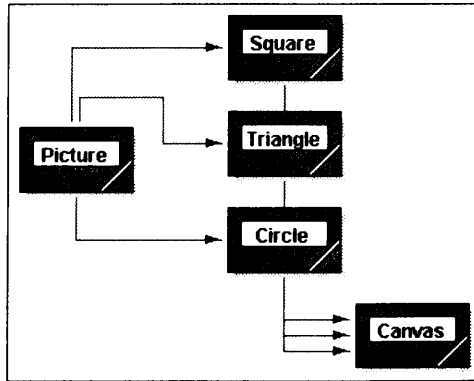
[그림 9-b] 객체 간 참조관계와 상태 변화 B



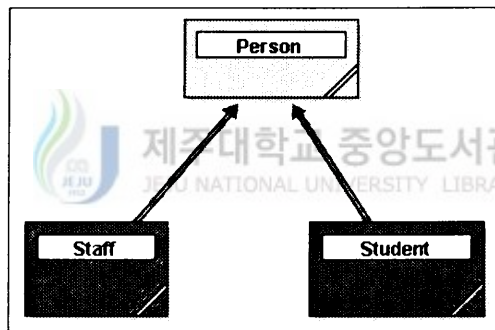
[그림 9-c] 객체 간 호출관계

2.3. 클래스 간 관계의 시각화

BlueJ에서 클래스의 관계를 시각화하기 위해 클래스 아이콘의 크기 조정 핸들과 클래스 아이콘의 이동은 화살표의 종류와 방향, 길이, 이동과 함께 패키지나 프로젝트의 레이아웃을 조정할 수 있다. 화살표가 싱글인 경우 클래스 간의 참조 관계를 나타내고[그림 10-a], 화살표가 더블인 경우는 클래스 간의 상속 관계를 나타낸다. 각각 다른 클래스의 색깔로 클래스의 유형을 구분하고 있다[그림 10-b]. 일반적으로 클래스 간 관계는 상속과 참조 관계가 같이 포함되어 있는데 이것은 위에서 제시한 [그림 6-a]에서의 클래스의 관계와 같이 시각화될 수 있다.



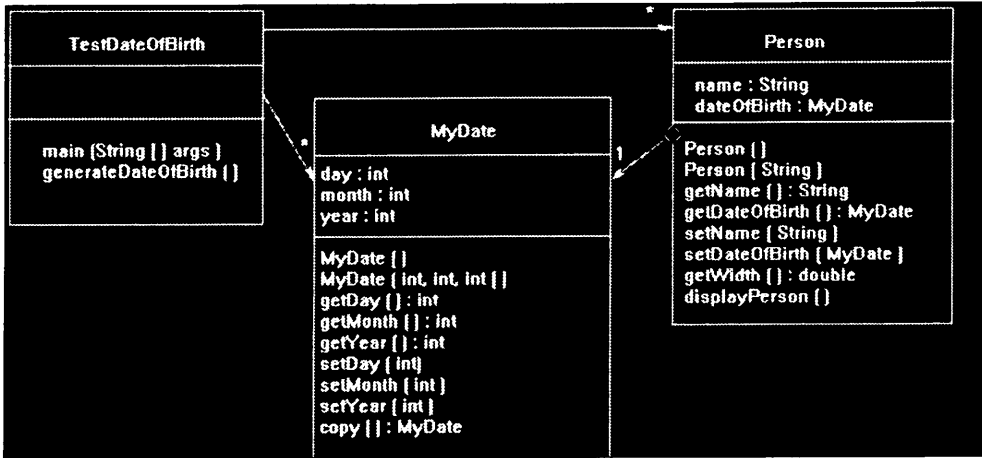
[그림 10-a] 클래스 간 참조관계



[그림 10-b] 클래스 간 상속관계

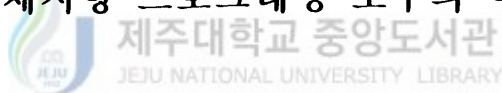
Java Demonstrations는 [그림 11-a]와 같이 클래스 간의 관계는 4)UML 식으로 시각화된다.

4) Unified Meta Language로 클래스들을 나타내는 다이어그램



[그림 11-a] 클래스 간 참조관계

3. 시각적 객체지향 프로그래밍 도구의 이용 방안



프로그래밍에서 초보자들을 어렵게 하는 이유 중 하나는 개념적으로 문제를 해결하는 것과 컴퓨터를 매체로 문제를 해결하는 것의 차이 때문이라고 할 수 있다. 프로그래밍 언어가 제공하는 개념은 컴퓨터의 일반적인 특성에 의존하므로, 높은 수준의 추상화가 요구된다. 초보자들이 프로그래밍 언어를 이해하여 그들의 생각을 정해진 형식대로 표현한다는 것은 어려운 일이 될 수 있으며[정 란95], 이와 같은 관점에서 프로그래밍 학습 초보자를 지원하기 위한 방안이 모색되어야 할 것이다. 또한 이와 관련하여 객체지향 프로그래밍 학습의 어려움을 인식하고 이를 지원하기 위한 시각적 도구의 개발과 이용을 검토하는 것은 중요한 일이라고 본다.

C언어를 이용하는 것과 같은 절차적인 프로그래밍에서는 코드와 실행 구조가 매우 직접적인 관계가 있어서 연산이 기대되는 곳에 필요한 코드를 쓸 수 있다. 그러나 객체지향 프로그래밍에서는 기대하는 연산의 직접적인 구현이 아니라 한 단계 위에서 추상화하여 클래스를 만들고 메소드를 호출

해야 기대되는 연산이 일어난다. 메소드가 실제적으로 수행시간 동안에 수행될 때, 특정 객체에 의해 소유되는 것이 시각화되어야하며 메소드를 쓸 때 인스턴스화된 객체의 관점으로부터 코드를 쓸 수 있어야 한다. 또한 메소드의 지역변수와 객체의 인스턴스 변수, 클래스 메소드의 클래스 변수는 혼동되기 쉽다[Clark&MacNIsh98]. 그러므로 객체지향 프로그래밍 구현 영역인 정적 영역과 실행 영역인 동적 영역의 연계부분을 시각화시키는 것이 중요하다. 이와 같이 관점에서 앞에서 비교·분석한 두 가지 시각적 도구의 이용 방안을 다음과 같이 제시한다.

첫째, 앞으로 진행될 학습의 기초가 되는 단계로서 객체지향 프로그래밍 학습자는 BlueJ를 통하여 객체와 클래스, 클래스 간 관계와 같은 객체지향 프로그래밍의 구조를 학습할 수 있다. 특히 클래스로부터 객체를 생성하고 그 객체의 메소드를 호출함으로써 발생하는 결과를 학습자와 BlueJ가 상호 작용적으로 학습할 수 있다.

둘째, 이전 단계에서 파악된 프로그래밍 개념을 바탕으로 학습자가 실제 적용하고 응용하는 단계로 BlueJ에 Java Demonstrations의 동일 애플리케이션을 입력으로 주어 실행해 보는 것이다. 이와 같은 과정을 통해 두 가지 도구에 나타난 객체지향 프로그래밍 개념의 표현 방식을 자연스럽게 비교하게 되고 실습하게 됨으로써 프로그래밍 학습의 부담이나 두려움 대신에 흥미를 가지고 프로그래밍 하게 될 것으로 기대한다.

셋째, 위에서 제시한 객체지향 프로그래밍의 기본 구조에 대한 학습을 바탕으로 Java Demonstrations를 이용하여 객체지향 프로그램 실행과정에서 객체의 동적인 변화 과정을 세부적으로 관찰하는 것이다. 이 과정을 통해 객체지향 프로그래밍에서 중요한 객체의 인스턴스 변수의 상태 변화 즉 객체의 상태변화와 인스턴스 메소드의 실행과정을 깊이 있게 보충하여 학습할 수 있어 BlueJ에서 학습한 객체지향 프로그래밍의 기본구조는 충분히

이해할 수 있을 것으로 기대한다.

넷째, 지금까지의 과정을 종합하는 심화된 학습을 하는 단계로 BlueJ에서 학습자가 프로그래밍한 임의의 프로그램에 대해 Java Demonstrations 식 표현 방식을 이용하여 나타내보고 토론하는 방안이다. 이와 같이 Java Demonstrations 방식 표현 자료를 가지고 학습자와 교수자, 학습자들 사이에서 서로 토론을 통하여 상호작용하는 동안 객체지향 프로그래밍 세계에 대해서 더 깊이 있게 이해할 수 있게 될 것이다.

지금까지 제시한 시각적 도구 외에도 Object Visualizer가 있으며 학습자는 이것을 통하여 상호작용적으로 특정 클래스의 객체들을 인스턴스화 할 수 있고 인스턴스화된 객체를 대상으로 인자들을 명시하여 특정 메소드를 호출함으로써 그 실행 결과를 볼 수 있도록 되어 있다. 하나의 클래스와 관련된 객체들을 다루어 볼 수 있을 뿐만 아니라 클래스와 클래스들 사이에서 객체들을 다루어 볼 수 있다. 즉 학습자 자신이 프로그래밍한 클래스와 다른 학습자가 프로그래밍한 클래스에서 생성된 객체들을 서로 인자로 전달할 수 있다. 이 도구는 객체지향 프로그래밍 개념을 소개하는 초급 단계에 특히 유용한 것으로 객체지향 프로그래밍 기초 단계에서 효과적으로 이용될 수 있다[Dershem&Vanderhde98].

V. 결론 및 제언

객체지향 프로그래밍을 교수·학습하는데 있어서 컴퓨터를 이용하여 프로그램이 어떻게 구성되고 실행되는지를 시각화함으로써 좀 더 동적이고 상호작용적인 교수·학습이 이루어질 수 있다. 이에 따라 본 연구는 객체지향 프로그래밍 기초 능력 배양을 위한 교수·학습 방법의 일환으로 객체지향 프로그래밍을 시각화한 도구들을 분석하고 그 이용 방안을 제시하였다. 이를 위해 우선 객체지향 프로그래밍의 이론적 배경에서 객체지향 프로그래밍의 배경과 특징, 단점과 장점에 대해서 살펴보았다. 이어서 객체지향 프로그래밍 교수·학습에서는 객체지향 프로그래밍 교육의 필요성과 객체지향 프로그래밍 교육에 대한 유의점을 관련 문헌을 통하여 정리하여 제시하였다. 그리고 객체지향 프로그래밍을 교수·학습을 위해 다루어야 될 객체지향 프로그래밍 관련 개념들을 객체와 클래스, 객체 간의 상호작용, 클래스 간 관계를 중심으로 정리하여 보았다. 이를 바탕으로 하여 객체지향 프로그래밍 시각적 도구들의 시각화 방법을 비교·분석하였고 그 이용 방안을 제시하였다.

객체지향 프로그래밍의 기초 능력 배양을 위한 교수·학습의 관점에서는 객체지향 프로그래밍의 유용성인 큰 프로그램에서의 코드의 재사용보다는 간단한 프로그램에서 객체지향 개념을 다루면서 학습을 시작해야 한다는 한계점이 있다. 그러나 중요한 것은 객체지향 프로그래밍 학습은 객체와 클래스에 대한 보다 확실한 이해를 기반으로 코드의 재사용이라는 상속성과 다형성으로 확대되어야 할 것으로 본다. 이와 같은 맥락에서 본 논문에서 제시하고 있는 시각적 도구의 분석과 이용 방안은 객체지향 프로그래밍 초보자들에게 객체와 클래스에 대한 핵심 개념을 강조하고 있어 객체지향

프로그래밍 기초 능력을 기르기 위한 지침이 될 수 있을 것으로 기대한다.

그런데 본 연구에서 제시하고 있는 두 가지 객체지향 프로그래밍을 위한 시각적 도구의 상호 보완적인 이용 방안을 고려해 볼 때, 앞으로 연구되어야 할 객체지향 프로그래밍 학습을 위한 시각적 도구는 이 두 가지 기능이 하나로 통합된 도구가 되어야 할 것이다. 즉 BlueJ에 Java Demonstrations와 같은 기능이 통합되어 있어서 학습자가 입력한 임의의 프로그램이 Java Demonstrations식의 실행 과정으로 시각화되도록 할 수 있어야 할 것이다. 또한 Java Demonstrations의 학습 내용을 단계별로 나누어 학습자의 수준에 맞게 선택할 수 있는 옵션 영역을 포함해야 할 것이다. 그리고 현재 Java Demonstrations는 특정하게 주어진 하나의 애플리케이션에 대한 실행 과정으로 고정되어 있다. 그러나 그것을 학습자가 학습하고자 하는 임의의 프로그램을 언제든지 입력으로 줄 수 있고 그 프로그램의 수행 과정을 출력으로 시각화시켜주는 일반화된 기능을 갖춘 상호작용적인 도구로 개발되어야 할 것이다.

참 고 문 헌

- [김형국99] 김형국, 주예찬, 문석원 공저, 블루엣으로 시작하는 자바 프로그래밍, 영진 출판사, 1999.
- [정 란95] 정 란, 초보자 프로그래밍 과정의 특성 분석과 지원 방안, 삼육대학교 논문집, 제29집, 1995.
- [우치수98] 우치수외, 프로그래밍 언어론, 이한 출판사, 1998.
- [이수경98] 이수경, 애니메이션과 인지양식이 과학적 이해와 파지에 미치는 영향, 교육공학연구 제14권 제2호, 1998.
- [유홍준97] 유홍준, 객체 지향 개념, 도서출판 홍은, 1997.
- [표창우98] 표창우, 프로그래밍 언어 교육 현황 및 개선 방향, 정보과학 학회지 제16권 제9호, 1998.
- [Armstrong99] Eric Armstrong, JBuilder 2 Bible, 교학사, 이관철 편역, 1999.
- [Allen00] Mark Allen, Data Structures and Problem Solving Using Java, Weiss Florida international University ADDISON-WESLEY, 2000.
- [Clark&MacNish&Royle98] David Clark, Cara MacNish, Gordon F. Royle, Java as a teaching language-opportunities, pitfalls and solutions, ACSE98, 1998.
- [Dershem&Vanderhde98] Herbert L. Dershem and james Vanderhde Java Class Visualization for Teaching Object-Oriented Concepts, ACM, SIGCSE, 1998.
- [Hadjerrouit99] Said Hadjerrouit, A Constructivist Approach to Objected-Oriented Design and Programming, ACM, ITiCSE, 1999.
- [Holand&Griffiths&Woodman97] Simon Holand, Robert Griffiths, Mark

- Woodman, Avoiding Object misconceptions, ACM, SIGCSE, 1997.
- [Hong98] Jason Hong, The Use of Java as an Introductory Programming Language, ACM Crossroads Student Magazine The ACM's First Electronic Publication, 1998.
- [Kolling&Rosenberg00] Michael Kolling and John Rosenberg, Objects first with Java and BlueJ (seminar session), Proceedings of the thirty-first SIGCSE technical symposium on Computer science education, 2000.
- [Lewis00] John Lewis, Myths about Object-Orientation and its Pedagogy, ACM, SIGCSE, 2000.
- [Madsen&Møller-Perderson88] Ole lehmann Madsen&BirgerMøller-Perderson, What objected-oriented programming may be and what is does not have to be, 1988.
- [Mayer&Gallin90] Mayer, R. E.& Gallini, J. K., When is an illustration worth ten thousand words?, Journal of Educational Psychology, 1990.
- [Mayer&Anderson91] Animations need narrations: An experimental test of a dual-coding hypotheses, Journal of Educational Psycholog, 1991.
- [Raner00] Mirko Ranger, Teaching Objet-Oriented with the Object Visualization and Annotation(OVAL), ACM, ITiCSE, 2000.

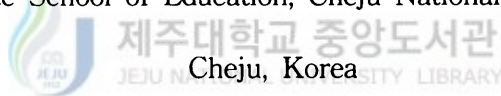
ABSTRACT

A Plan for Utilization of the Visual Tools To Cultivate the Basic Abilities of Object-Oriented Programming

Jeongmin Yang

Computer Education Major

Graduate School of Education, Cheju National University



Cheju, Korea

Supervised by Professor Cheolmin Kim

Visualization is one of the useful ways of effective teaching and learning that helps learners comprehend their learning objectives more deeply and swiftly. As computer technology is becoming cutting-edge, teaching and learning methods through visualization are being studied in various fields. Studies in visualization pertinent concepts, as well, are

* A thesis submitted to the Committee of the Graduate School of Education, Cheju National University in partial fulfillment of the requirements for the degree of Master of Education in August, 2001.

actively under way in the Object-Oriented Programming(OOP) areas which are becoming more important as the demands of software development and system design are on the rise. As a result of these studies, several visual materials were developed. However, the analysis and comparison of their functions and characteristics hasn't been demonstrated yet and the detailed plan for practical uses hasn't been provided. Therefore, the frequency of actual uses is not high.

This thesis covers some existing visual materials which were designed for OOP education, and analyzes their characteristics, and provides a detailed plan for utilization to cultivate the basic ability of OOP. The followings are summaries of this project.

First, the comprehensive contents for OOP such as a background of its introduction and its characteristics, merits & drawbacks, and kinds of OOP languages were identified on the basis of existing materials.

Second, some contents related to OOP which should be dealt with in teaching and learning process were systematically arranged. To begin with, this paper describes the need of OOP education and explains why the Java programming language was chosen as a teaching objective. It pointed out the difficulties of teaching and learning which teachers or learners might face due to the restricted concepts of specific OOP languages or misunderstandings of a program execution environment, and divided the main substances related to OOP into the three areas which were object and class; the interaction between objects; and the relation between classes.

Third, a plan was drawn for complementarily utilizing the existing tools which could be adapted to systematically cultivate the basic OOP abilities by comparing or analyzing the visualization methods of the visual tools for OOP education focusing on object and class, the

interaction between objects, and the relations between classes. Additionally, as an important prerequisite which makes the visual materials possible to cultivate the basic capabilities of OOP, the systematicness of selecting visual areas, the interaction with users, and the possibilities of selectional or level-oriented learning should be.

