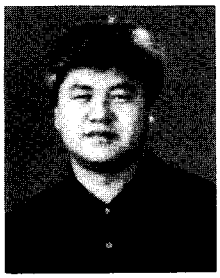
객체지향기술 : "소프트웨어 위기"를 해결하기 위한 해결책(I)

Object-Oriented Technology: A Silver Bullet for Software Crisis



임성택
고려대학교
경상대학 경영정보학과 조교수
Rim, seong-taek. Ph. D.
Assistant Professor.
Dept of Management Information Systems College of Economics & Commerce, Korea University

연재순서

1. 서론

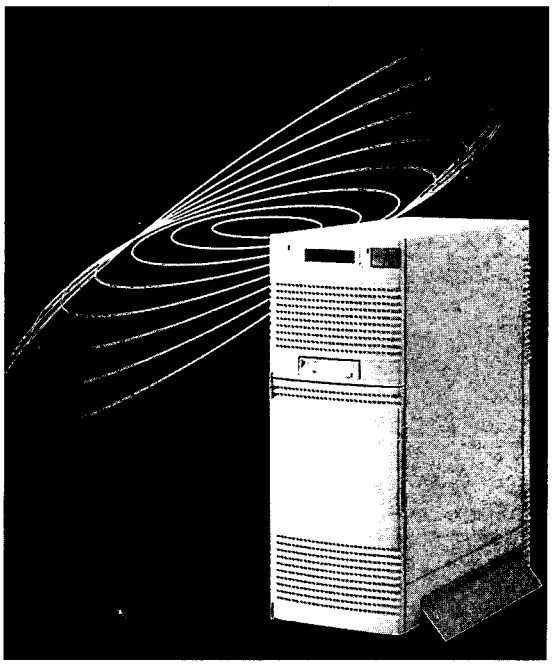
객체지향기술은 요즈음 소프트웨어 개발분야에서 많은 관심과 기대감을 불러 일으키고 있다. 심지어 객체지향기술을 적극적으로 지지하는 사람들은 객체지향기술을 '소프트웨어의 산업혁명'에까지 비유하고 있다. 하지만 '객체지향(Object-Oriented)'이 정확히 무엇을 의미하는 지에 대해서는 전문가들사이에서 조차 의견일치가 이루어지지 않고 있다. 이러한 상황에서 기업의 정보시스템 담당자나 일반관리자가 객체지향기술이 갖고 있는 잠재력을 정확하게 이해하고, 이를 기업에 적용한다는 것은 결코 쉬운 일이 아니다.

이 글의 목적은 기업의 정보시스템 담당자나 일반관리자들에게 객체지향기술을 소개하는 데 있다. 여러가지 측면에서 설명이 가능하겠지만 여기서는 객체지향기술을 기본개념, 객체지향 소프트웨어공학(객체지향 분석, 설계, 프로그래밍), 객체지향 데이터베이스의 세가지로 구분하여 소개하려 한다.

맨먼저, 객체지향기술의 기본개념에 대해 설명한다. 객체(Objects), 메시지(Message), 클라스(Class)의 세가지 핵심개념을 중심으로 캡슐화(Encapsulation), 정보은폐(Information

hiding), 상속(Inheritance), 다형성(Polymorphism), 동적제약(Dynamic Binding)등의 개념도 함께 소개한다.

그다음, 객체지향분석, 설계, 그리고 프로그래밍으로 이어지는 객체지향소프트웨어 공학에 대해서 살펴본다. 많은 사람들이 종래의 구조적인 패러다임(구조적 분석, 설계, 프로그래밍)에서 객체지향 패러다임으로 전환하는 것이 소프트웨어 위기를 극복할 수 있는 하나의 해결방안이라고 주장한다.



여기서는 이러한 주장의 근거에 대해서 분석해 본다. 마지막으로, 객체지향 데이터베이스에 대해서 살펴본다. 객체지향기술에서는 데이터베이스에 저장되는 것이 바로 객체이다. 여기서는 객체지향 데이터베이스란 과연 무엇이며, 객체지향 데이터베이스가 반드시 갖추어야될 기능은 무엇이며, 현재 시중에 나와있는 상업용 객체지향 DBMS들이 얼마나 이러한 기능들을 지원하는지를 알아본다.

2 객체지향의 기본개념

'객체지향(Object-Oriented)'이란 무엇을 의미하는지에 대해서는 사실 의견이 분분하다. 이번 절에서는 객체지향기술의 세가지 기본개념(객체, 메시지, 클래스)을 객체지향 프로그래밍에 촛점을 맞추어 살펴보기로 하자.

2.1 객체(Objects)

객체지향기술에서 가장 중요한 개념은 객체(Objects)이다. 프로그래밍 측면에서 객체는 서로 관련이 있는 절차(Procedures)와 데이터(Data)를 함께 수집해 논 소프트웨어 패키지이다. 객체지향기술에서는 절차를 메소드(Methods), 행위(Behavior), 또는 멤버함수(Member Function)라고도 하며, 데이터는 속성(Attributes), 변수(Variables), 상태(State) 또는 데이터 멤버(Data Member)라고도 한다. 이글에서는 각각 메소드와 속성으로 부르기로 하자.

속성은 객체의 상태(State)를 나타내기 위해 사용되고, 메소드는 객체의 행위(Behavior)를 나타내는데 사용된다. 서로 연관이 있는 데이터와 절차를 함께 묶는 것을 캡슐화(Encapsu-lation)라고 한다.

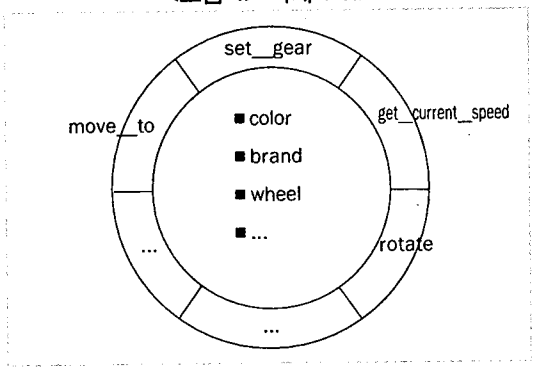
그리고 캡슐화된 항목들은 미리 명확하게 선언된 인터페이스만을 통해서만이 접근이 가능하다. 하지만 이러한 인터페이스는 이 객체가 할 수 있는 것만을 선언할 뿐 구체적으로 이것을 어떻게 실현하는지는 말해주지 않는다. 이

를 정보은닉(Information Hiding)이라 한다.

예를들어 자동차를 객체로 표현해보자. 자동차는 손님을 실어나른다든지, 기어를 작동한다 등의 여러가지 다양한 행위를 할 수 있고, 또한 색깔, 차종 등 여러가지 자체의 특색에 대한 정보를 가질 수 있다. 자동차를 객체로 표현하기 위해서는 그것의 행위는 메소드로, 자동차의 특색은 속성으로 정의할 수 있어야 한다.

따라서 자동차라는 객체는 moveto, set__current__speed 또는 set__gear와 같은 메소드와 color, brand등과 같은 속성을 가질 수 있다. 〈그림 1〉은 이러한 자동차라는 객체를 표현해 주고 있다.

<그림 1> 객체 : car



객체지향기술에서 객체의 속성은 객체의 메소드를 통해서만 접근이 가능하다. 이것이 〈그림 1〉에서 객체를 메소드들로 구성되어 있는 외부의 동심원이 내부의 속성들을 감싸는 형식의 그림으로 표현한 이유이다.

set__gear라는 메소드는 car라는 객체의 현 상태를 변경시킬 수 있는 메소드의 예이며, get__current__speed라는 메소드는 이 객체의 상태를 읽기 만하는 메소드이다.

객체라는 개념은 간단해 보이지만 아주 강력하다. 객체들은 서로서로 독립적으로 정의되고 유지되기 때문에 아주 이상적인 소프트웨어 모듈이라고 할 수 있다. 객체가 아는 것은 모두 속성의 값(value)으로서 표현되어 있고, 객체가 할 수 있는 모든 것은 메소드에 의해서 표현된다.

2.2 메시지

객체가 아무리 잘 만들어졌다 하더라도 그것이 고립되어 있다면 아무런 쓸모가 없다. 객체의 가치는 다른 객체들과의 교류에서 찾을 수 있다. 객체들간의 교류는 메시지(Message)를 통하여 이루어진다.

전통적인 프로그래밍의 측면에서 보면 메시지란 함수를 부르는 것(Function Call)과 같다.

일반적으로 메시지는 받는 객체의 이름(Receptor), 수행되어야 할 메소드의 이름(Selector), 메소드에 의해 요구되는 변수나 값(Arguments)의 세부분으로 구성되어 있다. 메시지를 보내는 객체를 클라이언트(Client)이라 하고, 받는 쪽을 서버(Server)라 한다. 필요하다면 서버도 다른 객체에 메시지를 보낼 수 있다. 메시지가 쓰여지는 방법은 사용하는 프로그래밍언어에 따라 다르다.

예를들면 C++에서 메시지는 다음과 같을 수 있다.

```
car104.rotate(90).
```

여기서 car104는 메시지를 받는 객체의 이름이고, rotate는 수행하여야될 메소드를 나타낸다. 이 rotate 메소드는 하나의 인자를 요구하는 데 이 경우에는 회전각도를 필요로 한다.

다른 객체에서 한 객체의 상태를 직접 접근할 수 없다는 것이 불편할 수도 있지만, 그러나 바로 이러한 특성이 프로그래밍전략중의 하나인 정보은폐(Information Hiding)를 강력하게 지원한다.

다시말해 객체의 속성은 그 객체에서만 사용이 가능하고, 시스템의 나머지 부분들로부터는 감추어질 수 있다. 각 속성들의 값(Value)은 메소드를 선언함으로써만 접근이 가능하다.

객체가 메시지를 받았을 때 어떤 메소드를 실행시킬 것인가를 결정하는 과정을 제약(Binding)이라 한다.

제약에는 정적제약(Static Binding)과 동적제약(Dynamic Binding) 두가지가 있다. 정적제약이란 수행되어야할 메소드가 컴파일될 때(Compile Time) 선택된다는 것을 의미하고, 동적제약이란 수행되어야할 메소드가 실행될 때(Run Time) 결정되는 것을 말한다.

동적제약을 필요로 하는 경우는 각기 다른 객체들에 똑같은 이름을 가진 메소드가 존재할 경우이다.

이러한 기능을 다형성(Polymorphism)이라 한다. 다형성이 주는 잇점은 객체들을 좀더 서로 독립적으로 만들고 최소한의 변화만으로 새로운 객체를 추가할 수 있다는 것이다. 예를들어 여러가지 형태의 모양을 그려야 되는 그림그리기 프로그램을 생각해 보자.

전통적인 프로그래밍 언어에서는 각각의 모양을 그리기위한 명령어가 각각 독특한 이름 예를들면 DrawPoint, DrawLine, DrawRectangle 등을 가져야 한다.

하지만 객체지향프로그래밍에서는 삼각형, 원, 사각형등의 모양은 서로 다른 클래스에 의

해서 만들어진다. 각 클래스에 있는 메소드가 하는 기능은 조금씩 다르지만, 이들 메소드들의 이름은 하나로 예를들면 draw와 같이 될 수 있다. 각 클래스는 자신이 갖고 있는 버전의 draw에 대해서만 알기 때문에 프로그램내에서 혼동될 이유가 없다.

이러한 기능을 프로그래밍에서는 오버로딩(Overloading)이라 한다.

2.3 클래스

'메시지를 통해서 서로 교류하는 객체'라는 개념은 객체지향기술의 핵심을 이룬다. 하지만 추가적으로 클래스라는 개념은 현실을 아주 효율적으로 표현할 수 있도록 해준다.

먼저 클래스와 인스턴스(Instance)의 관계를 살펴보자. 클래스란 일정 그룹의 객체들에 대해 메소드와 속성을 정의하는 틀(Templeate)이다. 일단 한번 클래스가 정의되면, 그 클래스는 원하는 만큼의 인스턴스를 만들어낼 수가

있다. 프로그래밍의 측면에서 보면 클래스란 사용자가 정의한 데이터 타입이라고 할 수 있고, 인스턴스는 이러한 데이터 타입을 선언하는 것이다.

클래스는 모든 인스턴스에 의해 공유하는 특색(메소드와 속성)을 정의하지만, 인스턴스는 그들을 구별할 수 있게 만드는 그들 자신 고유의 정보를 지닐 수 있다. 여기서 그들 자신의 고유한 정보란 그 인스턴스들의 상태들이 각기 다름을 의미한다.

즉 한 클래스에 속하는 객체는 클래스에서 정의된 속성에 구체적인 값(상태)를 갖게 된다. 클래스와 인스턴스로서의 객체를 구별짓는 것은 속성의 값이다. 실제로 인스턴스는 값을 지닐 수 있는 일련의 변수만으로 이루어졌다. 인스턴스가 메시지를 받았을 때, 그 인스턴스는 그것의 메소드와 속성(변수의 타입)을 클래스에서 참조해, 이들 속성에 각각의 값을 적용한다.

예를 들면 자동차라고 하면 연상되는 것은 일반적으로 말하는 자동차와 '우리집에 있는 콩코드 7285' 처럼 구체적인 자동차로 구분해 볼 수 있다. 일반적으로 말하는 자동차는 car라는 클래스를 의미한다고 할 수 있고, 구체적인 자동차는 이 car라는 클래스의 인스턴스를 의미한다고 할 수 있다.

따라서 car라는 클래스의 속성중의 하나가 차종이고, 우리집 콩코드가 이 클래스의 인스턴스라면 그 차종이라는 속성은 '콩코드'라는 값을 갖게 된다. 다음으로 클래스들간의 관계에 대해서 살펴보자. 일반화(Generalization)/구체화(Specialization)는 클래스들의 계층(Hier-achy)을 만드는 수단을 제공한다.

현존하는 클래스들의 공통점들만을 상위클래스(Superclass)에서 잡아낼 수가 있다. 이러한 계층을 만드는 이유는 상속(Inheritance)에서 찾을 수 있다.

상속이란 어떤 클래스는 그것보다 더 일반적인 클래스의 특수한 경우라고 생각할 수 있고, 따라서 이 클래스는 일반 클래스가 갖는 메소드와 속성을 자동적으로 갖는 메카니즘을 말한다. 기본생각은 상위클래스(Superclass)에 타당한 메소드와 속성은 하위클래스(Subclass)에도 타당하다는 것이다.

그러므로 하위클래스는 상위클래스의 모든 메소드와 속성을 상속할 수 있고, 또한 하위클래스에 고유한 메소드와 속성을 추가할 수도 있다. 예를들어 car 클래스의 경우에서 스포츠카를 추가하고 싶다고 하자. 이경우에는 색깔이나 차종외에도 최고속도에 관심이 있을 것이다.

따라서 이 클래스는 car 클래스의 모든 메소드와 속성을 상속하고, 추가로 top__speed라는 속성와 get__speed라는 메소드를 이 하위클래스에서 정의할 수 있다. 그러면 객체가 어떻게 메소드나 속성을 찾는지 살펴보자.

예를들면 자동차란 객체가 다음의 메시지를 받았다고 가정하자: car104.set__gear(2). car104란 객체는 car라는 클래스에서 set__gear란 메소드를 찾고, 그 메소드를 실행시킨다. 다음 set__gear란 메소드는 기어의 상태를 2라는 값으로 변경시킨다.

만약 객체가 그것이 속해있는 클래스에 없는 메소드를 수행하도록 메시지를 받았을 때 그 객체는 자동적으로 그 메소드를 발견하기 위해 클래스 계층구조(Class Hierachy)를 따라 그 메소드를 찾게 된다. 속성을 찾기 위해서도 마찬가지이다. *DC*