**객체지향 문법 정리 2015. 04. 29**

- 절차 지향과 객체 지향

절차 지향의 핵심은 데이터를 중심으로 구현을 한다는 데 있다. 각각의 기능을 수행하는 프로시저(함수)들이 데이터를 조작하거나 사용하는 방식으로 구현을 한다.

다양한 프로시저들이 데이터를 공유해서 동작하기 때문에 데이터의 변화는 곧 데이터를 사용하는 코드(프로시저)의 변화로 이어진다. 심지어 한 데이터 타입의 변화가 다른 데이터의 변화를 일으키기는 경우도 있다. 예를 들어, 화폐를 표현하는 데이터의 타입이 int에서 double로 변경되면 화폐를 사용하는 프로시저 뿐만 아니라 잔고, 판매금액 등의 다른 데이터 타입까지도 모두 변하게 된다. 심할 경우, 한 변수의 타입 변화가 수 일이 소용되는 노가다를 만들어내기도 한다. 이는 데이터 중심인 절차 지향으로 구현할 경우 유지보수 비용이 높아진다는 것을 의미한다.

중심에 데이터를 놓고 프로시저를 구현하는 절차 지향과 달리 객체 지향은 데이터와 기능을 함께 담고 있는 객체들이 서로 대화하는 방법으로 구현된다. 객체 간에는 데이터를 (최대한) 주고 받지 않으며, 객체 상호간에 기능을 실행해 달라고 요청하는 방식으로 구현을 하게 된다.

각 객체들은 내부적으로 데이터를 담고 있지만 다른 객체에게 데이터를 노출하지는 않는다. 객체와 객체 간에 데이터가 흐르지 않기 때문에 특정 객체가 담고 있는 데이터에 변화가 생기더라도 해당 객체로만 변화가 한정되며 다른 객체에까지 변하가 전파되지 않게 된다. 이 얘기는 변화에 따른 유지보수 비용이 절차 지향에 비해 상대적으로 더 작다는 것을 의미한다.

(출처: <http://javacan.tistory.com/193> (Java Can Do It))

1. 캡슐화 (Encapsulation)

1.1 캡슐화란?

객체지향에서 캡슐화라는 개념은 클래스 내부에 여러 속성과 여러 오퍼레이션을 함께 묶음을 의미한다. 그리고 캡슐화는 클래스 내부의 속성이나 오퍼레이션을 외부에 노출하지 않고 보호하는 것을 의미한다. 이렇게 캡슐화는 묶는 것과 보호하는 것을 생각할 수 있다. 좀 더 상세하게 생각해 보면, 여러 속성과 여러 오퍼레이션을 함께 묶어 클래스로 취급하는 것과 클래스 내부를 외부에서 접근하지 못하도록 보호하는 것이 바로 캡슐화이다. 즉, 속성과 오퍼레이션을 하나의 클래스로 패킹한 것이 바로 캡슐화라는 개념이다. 외부로부터 내부를 감싸 숨기는 것을 캡슐화라고 한다.

1.2 캡슐화는 왜 해야 할까?

캡슐화를 통해 묶고 숨김을 생각해 보았다. 그런데 왜 묶어야 하고 숨겨야 하는 것일까? 먼저, 묶음으로 인해 프로그램을 바라보는 단위가 커진다. 이전의 프로그래밍 언어인 C언어는 프로그램을 함수 단위로 구조화할 수 있으나, 프로그램 소스가 커지면 이해하기 어렵고 관리가 힘들어 질 수 있었다. 그러나 객체지향 프로그램에서는 프로그램 소스를 클래스 단위로 바라보게 됨으로써 좀더 복잡하고 커다란 소스코드도 쉽게 이해하게 되었다. 왜냐하면 클래스 내부에 여러 함수를 내포할 수 있기 때문에 프로그램 소스 코드를 바라보는 단위가 커졌으며, 그로 인해 프로그램 관리가 좀 더 수월해진 것이다.

두 번째, 내부를 숨김으로써 내부를 좀더 자유롭게 변경할 수 있게 되었다 이전의 함수 중심적인 구조적 프로그래밍 언어에서는 프로그램 내부에서 데이터가 어디서 어떻게 변경되는지 파악하기 어려웠고, 그로 인해 유지 보수가 힘들었기 때문에 자료를 중심으로 함수가 종속되는 구조가 되기도 하였다. 객체 지향에서는 클래스 내부의 데이터를 외부에서 참조하지 못하도록 차단하여 이러한 폐단을 없앨 수 있다. 이렇게 내부의 데이터나 함수를 외부에서 참조하지 못하도록 차단하는 개념을 정보 은닉Information Hiding)이라고 하며 이것이 바로 캡슐화라는 개념이다.

2. 추상화 (Abstraction)

추상화란 무엇일까?

객체지향에서 추상화라는 개념은 '객체에서 공통된 속성과 행위를 추출하는 것'을 의미한다.

예를 들어 홍길동 교수, 이순신 교수, 강감찬 교수가 있다고 하자. 이 교수들을 객체로 보고 공통된 속성과 오퍼레이션으로 '교수'라는 클래스를 정의한다고 해보자. 공통된 속성으로 이름, 주민등록번호, 강의 분야, 주소, 전화번호를 추출하고, 공통된 행위로써 '강의하다','채점하다','과정 등록하다' 를 추출했다고 하면, 그를 포함한 클래스가 생성된다.

[객체기술사전]에 따르면, 추상화란 '중요하지 않거나, 주 관심 대상이 아닌 자세한 부분은 감추거나 무시하고, 가장 중요하고, 근간이 되고, 다른 대상들과 구분될 수 있는 면만을 포함하고 있는 모델이며, 공통점을 강조하기 위해 차이점을 제거한 결과물'이라고 정의하고 있다. 실제로 객체의 속성과 행위 중에서 관심 대상이 아닌 부분은 드러낼 필요가 없는 것이 추상화이다.

즉, 추상화는 다른 객체들과 구분되는 핵심적인 특징들에만 집중함으로써, 복잡도를 관리할 수 있도록 한다. 주의할 점은 추상화는 문제 영역과 관점에 의존적이라는 것이다. 그래서 어떤 영역에서 중요한 것이 다른 영역에서는 그렇지 않을 수도 있다. 예를 들어 학생이라는 클래스와 멀티캠퍼스와 같은 단기 과정을 수강하는 곳의 시스템 내부에서 모델링되는 학생이란 클래스는 내부의 속성과 오퍼레이션이 서로 많이 다를 것이다. 따라서, 하나의 대상에 대하여 목적이나 원하는 기능에 따라 여러 추상화 모델이 생성될 수 있다.

3. 상속 (Inheritance)

3.1 상속이란?

상속의 사전적 의미는 자신이 가지고 있는 것을 하위에게 물려주거나, 하위에서 물려받는 것이다. 객체지향에 상속도 마찬가지 의미이며, 클래스의 속성과 오퍼레이션을 하위 클래스에 물려주거나, 상위클래스에서 물려받는 것을 지칭한다. 예를 들어 자바 API에서 최상위 클래스인 Object 클래스를 생각해보자.

Object 클래스 내부에는 다음과 같은 메소드가 존재한다.

Protected void finalize()

Class getClass()

int hashCaode()

void notify()

void notifyAll()

String toString()

void wait()

void wait(long timeout)

void wait(long timeout, int nanos)

자바의 모든 클래스는 Object 클래스의 하위 클래스가 되므로 어떤 클래스에서든 Object 클래스에 정의된 위와 같은 메소드를 사용할 수 있다. 이와 같이 상속이란 하위에게 사용할 수 있도록 물려주는 것이다.

3.2 상속을 사용하여 좋아지는 것은?

먼저, 재사용으로 인해 코드가 줄어든다. 하위 클래스에서 속성이나 오퍼레이션을 다시 정의하지 않고 상속받아서 사용함으로써 코드가 줄어든다. 그리고 좀 더 범용성있게 사용할 수 있다.

참고로 하위 클래스는 상위 클래스가 가지고 있는 모든 자료와 메소드를 물려받아 자유롭게 사용할 수 있지만, 또한 자신만의 자료와 메소드를 추가적으로 덧붙임으로써 새로운 형태의 클래스로 발전하게 된다는 것도 잊지 말자.

3.3 상위 클래스와 하위 클래스의 생성자

생성자라는 것은 일종의 오퍼레이션이며, 클래스가 객체화될 때 실행된다. 그런데 하위 클래스의 생성자를 호출할 경우, 자동적으로 상위의 생성자가 호출됨을 반드시 기억해야 한다

하위의 생성자가 호출될 때 묵시적으로 상위의 생성자를 자동으로 호출하는 것이다.

4. 다형성(Polymorphism)

﻿4.1 다형성(Polymorphism)이란?

객체지향 개념에서 가장 중요한 것이 바로 다형성이다. 다형성이라는 개념은 객체 지향 개념의 꽃이라고 할 수 있을 정도로 중요하고 근사한 개념이다. 웹사전에 따르면, 다형성의 일반적인 의미는 '다양한 형태로 나타날 수 있는 능력'이라고 한다. 또 다른 웹 사전을 보면, 객체지향에서의 다형성은 '여러 클래스들이 동일한 이름의 오퍼레이션을 서비스하도록 하는 것'이라고 한다.

이러한 다형성은 실제의 코드에서는 '하나의 클래스 내부에 같은 이름의 오퍼레이션을 여럿 정의하거나, 상위 클래스의 오퍼레이션을 하위 클래스에서 다시 정의함'으로써 구현한다.

예를 들어 상위클래스인 Object클래스에 toString(), equals() 메소드가 존재하지만, 하위 클래스인 String과 Date 클래스에도 toString(), equals() 메소드가 존재함으로 볼 수 있다. 이렇게 상위 클래스에 있고 상속받았으나 하위 클래스에서 다시 정의하는 것을 메소드 오버라이딩(Method Overriding)이라고 하며, 메소드 오버라이딩이 다형성이다.﻿ ﻿그리고 동일한 이름의 오퍼레이션이 여러개 정의되어 있는데 단지, 매개변수의 타입에 따라 서로 구분될 수 있다. 이렇게 클래스 내부에 동일한 이름의 오퍼레이션을 여럿 정의하는 것이 바로 메소드 오버로딩(Method Overloading)이라고 하는 다형성이다.

(출처: http://searchstory.tistory.com/entry/객체지향-기본-개념)