**객체지향이란?**

객체 지향 프로그래밍(영어: Object-Oriented Programming, OOP)은 컴퓨터 프로그래밍의 패러다임의 하나이다. 객체 지향 프로그래밍은 컴퓨터 프로그램을 명령어의 목록으로 보는 시각에서 벗어나 여러 개의 독립된 단위, 즉 "객체"들의 모임으로 파악하고자 하는 것이다. 각각의 객체는 메시지를 주고받고, 데이터를 처리할 수 있다.

객체 지향 프로그래밍은 프로그램을 유연하고 변경이 용이하게 만들기 때문에 대규모 소프트웨어 개발에 많이 사용된다. 또한 프로그래밍을 더 배우기 쉽게 하고 소프트웨어 개발과 보수를 간편하게 하며, 보다 직관적인 코드 분석을 가능하게 하는 장점을 갖고 있다. 그러나 지나친 프로그램의 객체화 경향은 실제 세계의 모습을 그대로 반영하지 못한다는 비판을 받기도 한다.

1990년대 중반 이후로 각광받고 있는 객체 지향 언어는 자바로 가전 제품에 사용될 소프트웨어의 개발 목적으로 썬 마이크로시스템즈의 제임스 고슬링에 의하여 고안된 언어이다. 1993년 고슬링은 월드 와이드 웹에 자바 언어를 적용할 것을 결정하면서 핫자바라는 웹 브라우저를 개발하였고, 이는 1995년 이후 넷스케이프사 쪽에서 지원을 받게 되었다. 자바 언어의 장점은 언어의 단순성과 플랫폼 독립성이다. 특히 언어의 단순성 입장에서 객체 지향 패러다임에 충실하게 언어가 고안되었기 때문에 C++보다 오용의 소지가 다소 적다.

**기본 구성 요소**

**클래스(Class)** - 같은 종류(또는 문제 해결을 위한)의 집단에 속하는 속성(attribute)과 행위(behavior)를 정의한 것으로 객체지향 프로그램의 기본적인 사용자 정의 데이터형(user define data type)이라고 할 수 있다. 클래스는 프로그래머가 아니지만 해결해야 할 문제가 속하는 영역에 종사하는 사람이라면 사용할 수 있고, 다른 클래스 또는 외부 요소와 독립적으로 디자인하여야 한다.

Ex) public class className{ … }

**객체(Object)** - 클래스의 인스턴스(실제로 메모리상에 할당된 것)이다. 객체는 자신 고유의 속성(attribute)을 가지며 클래스에서 정의한 행위(behavior)를 수행할 수 있다. 객체의 행위는 클래스에 정의된 행위에 대한 정의를 공유함으로써 메모리를 경제적으로 사용한다.

Ex) ObjectType variableName = new ObjectType();

**메서드(Method), 메시지(Message)** - 클래스로부터 생성된 객체를 사용하는 방법으로서 객체에 명령을 내리는 메시지라 할 수 있다. 메서드는 한 객체의 서브루틴(subroutine) 형태로 객체의 속성을 조작하는 데 사용된다. 또 객체 간의 통신은 메시지를 통해 이루어진다.

Ex) 접근제어자 리턴타입 메서드이름 (참조값) { 코드 블록 };

Public void main(String[] args){ … };

특징

객체 지향 프로그래밍은 기본적으로 자료 추상화, 상속, 다형 개념, 동적 바인딩 등이 있으며 추가적으로 다중 상속 등의 특징이 존재한다. 객체 지향 프로그래밍은 자료 추상화를 기초로 하여 상속, 다형 개념, 동적 바인딩이 시스템의 복잡성을 제어하기 위해 서로 맞물려 기능하는 것이다.

**자료 추상화**

자료 추상화는 불필요한 정보는 숨기고 중요한 정보만을 표현함으로써 프로그램을 간단히 만드는 것이다. 자료 추상화를 통해 정의된 자료형을 추상 자료형이라고 한다. 추상 자료형은 자료형의 자료 표현과 자료형의 연산을 **캡슐화**한 것으로 **접근 제어**를 통해서 자료형의 정보를 은닉할 수 있다. 객체 지향 프로그래밍에서 일반적으로 추상 자료형을 클래스, 추상 자료형의 인스턴스를 객체, 추상 자료형에서 정의된 연산을 메소드, 메소드의 호출을 메시지라고 한다.

Ex) public, private, protected, default

**상속**

상속은 새로운 클래스가 기존의 클래스의 자료와 연산을 이용할 수 있게 하는 기능이다. 상속을 받는 새로운 클래스를 부클래스, 파생 클래스, 하위 클래스, 자식 클래스라고 하며 새로운 클래스가 상속하는 기존의 클래스를 기반 클래스, 상위 클래스, 부모 클래스라고 한다. 상속을 통해서 기존의 클래스를 상속받은 하위 클래스를 이용해 프로그램의 요구에 맞추어 클래스를 수정할 수 있고 클래스 간의 종속 관계를 형성함으로써 객체를 조직화할 수 있다.

Ex) public className extends(or implements) className2() { … }

**다중 상속**

다중 상속은 클래스가 2개 이상의 클래스로부터 상속받을 수 있게 하는 기능이다. 클래스들의 기능이 동시에 필요할 때 용이하나 클래스의 상속 관계에 혼란을 줄 수 있고(예: 다이아몬드 상속) 프로그래밍 언어에 따라 사용 가능 유무가 다르므로 주의해서 사용해야 한다. **JAVA는 지원하지 않는다.**

**다형성**

다형성 개념이란 어떤 한 요소에 여러 개념을 넣어 놓는 것으로 일반적으로 **오버라이딩**(같은 이름의 메소드가 여러 클래스에서 다른 기능을 하는 것)이나 **오버로딩**(같은 이름의 메소드가 인자의 갯수나 자료형에 따라서 다른 기능을 하는 것)을 의미한다. 다형 개념을 통해서 프로그램 안의 객체 간의 관계를 조직적으로 나타낼 수 있다.

**동적 바인딩**

동적 바인딩은 실행 시간 중에 일어나거나 실행 과정에서 변경될 수 있는 바인딩으로 컴파일 시간에 완료되어 변화하지 않는 정적 바인딩과 대비되는 개념이다. 동적 바인딩은 프로그램의 한 개체나 기호를 실행 과정에 여러 속성이나 연산에 바인딩함으로써 다형 개념을 실현한다.

**장점**

소프트웨어 공학의 관점에서 볼 때 S/W의 질을 향상하기 위해 강한 응집력(Strong Cohesion)과 약한 결합력(Weak Coupling)을 지향해야 하는데, 객체 지향 프로그래밍의 경우 클래스에 하나의 문제 해결을 위한 데이터를 모아 놓은 데이터형을 사용함으로써 응집력을 강화하고, 클래스간에 독립적으로 디자인함으로써 결합력을 약하게 할 수 있다.