

객체 지향 프로그래밍

프로그래밍 패러다임

- [행위](#)
- [에이전트 지향](#)
- [배열 지향](#)
- [오토마타 기반](#)
- [병행 컴퓨팅](#)
 - [상대주의적 프로그래밍](#)
- [데이터 기반](#)
- [선언형](#) (반의어: [명령형](#))
 - [함수형](#)
 - [함수 논리형](#)
 - [순수 함수형](#)
 - [논리형](#)
 - [가추법](#)
 - [Answer set](#)
 - [병행 논리](#)
 - [함수 논리](#)
 - [귀납적 논리](#)
 - [제약](#)
 - [제약 논리](#)
 - [병행 제약 논리](#)
 - [데이터플로](#)
 - [플로 기반](#)
 - [반응형](#)
- [동적/스크립트](#)
- [사건 기반](#)
- [함수 레벨](#) (반의어: [값 레벨](#))
 - [포인트 프리 스타일](#)
 - [연결식](#)
- [제네릭](#)
- [명령형](#) (반의어: [선언형](#))
 - [절차적](#)
 - [객체 지향](#)
- [문학적](#)
- [언어 지향](#)
 - [자연어 프로그래밍](#)
 - [규율 지향](#)
 - [도메인 특화](#)
 - [문법 지향](#)
 - [의도적](#)
- [메타프로그래밍](#)
 - [자동](#)
 - [귀납적 프로그래밍](#)
 - [반영](#)
 - [속성 지향](#)
 - [매크로](#)
 - [템플릿](#)
- [비구조적](#) (반의어: [구조적](#))

- [배열](#)
- [비결정](#)
- [병렬 컴퓨팅](#)
 - [프로세스 지향](#)
- [확률론적](#)
- [스택 지향](#)
- [구조적](#) (반의어: [비구조적](#))
 - [블록 구조](#)
 - [객체 지향](#)
 - [행위자 기반](#)
 - [클래스 기반](#)
 - [병행](#)
 - [프로토타입 기반](#)
 - [관심사의 분리](#)에 따라:
 - [관점 지향](#)
 - [역할 지향](#)
 - [주체 지향](#)
- [재귀](#)
- [심볼릭](#)
- [값 레벨](#) (반의어: [함수 레벨](#))
- [퀀텀 프로그래밍](#)

ⓘ· 📄 🗺 🔗 (https://ko.wikipedia.org/w/index.php?title=%ED%8B%80:%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D_%ED%8C%A8%EB%9F%AC)

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


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

역사

객체 지향 언어의 시초


객체 지향 언어의 시초는 1960년 노위지안 컴퓨팅 센터의 조한 달과 크리스틴이 발표한 시물라67이다. 시물라67이 채택하고 있는 가장 중요한 개념은 클래스의 도입으로서 이 아이디어는 **스몰토크**, C++ 등에도 사용되었다. 하지만 시물라 67의 발표 이후 10여년 간 객체 지향 언어는 전혀 주목을 받지 못하였다. 1970년대 컴퓨터 산업을 주도한 IBM, AT&T, 미 국방성 등에서 관심을 두지 않았기 때문에 시물라 67은 실용적인 언어로 발전하지는 못하였다. 하지만 이의 학문적 가치는 인정받고 있다.

스몰토크

 [스몰토크](#) 문서를 참고하십시오.

객체 지향 언어로서의 실질적 원조는 제록스 기업의 팰러앨토 연구소에서 앨런 케이의 책임 하에 만들어진 스몰토크이다. 이 언어 역시 아이디어는 시물라 67에서 얻어왔지만 가장 순수한 객체 지향 언어로 만들어졌으며 현재에도 인정받고 있다. 미국에서 많은 사용자들을 확보하고 있다. 1970년대 말 스몰토크가 만들어질 당시 제록스에서는 3가지 가정을 하고 이 가정에 초점을 맞추어 스몰토크의 문법과 의미를 정의하였다. 첫 번째는 전 세계의 모든 사람이 컴퓨터를 사용할 것이라는 가정이었고, 두 번째는 모든 사용자가 그래픽이 지원되는 모니터와 마우스를 기본 설비로 사용하며 윈도 환경에서 작업할 것이라는 가정이었다. 마지막으로 모든 사람이 각자의 응용 프로그램을 쉽게 개발할 수 있어야 한다는 것이었다. 첫 번째와 두 번째 가정은 현실에서 거의 사실화되었으나 마지막 가정은 제록스의 실수였다. 현재 많은 컴퓨터 사용자들은 그들의 응용 프로그램을 스스로 개발하지 않는다. 이러한 점 때문에 스몰토크의 순수성과 독창성에 비하여 크게 성공하진 못하였다.

에이다

 [에이다 \(프로그래밍 언어\)](#) 문서를 참고하십시오.

에이다는 1980년대 초 객체 지향 프로그래밍 언어로 미 국방성에서 개발한 것이다. 미 국방성은 에이다 개발 전까지 코볼과 포트란을 이용하여 시스템을 개발하였는데 프로젝트 규모가 점점 커져 가면서 그것의 유지와 보수 비용의 문제가 따랐다. 이 문제를 해결하기 위하여 코볼과 포트란의 환경의 개발을 시도하였으나, 한계를 느끼고 새로운 언어를 도입하게 된다. 미 국방성은 새로운 언어에 대한 정의를 공모하였으며, 여러 업체들이 제시한 언어들을 기준으로 에이다를 정의하였다. 그 당시에는 파스칼이 인기가 좋아서 에이다의 문법은 기본적으로 파스칼의 문법을 기반으로 하였다.

에이다의 큰 특징은 예외 처리 기능의 도입이다. 이는 시스템의 신뢰도를 높이기 위한 중요한 기능이며, 미 국방성 프로젝트가 중요시 하는 신뢰도를 증가시키기 위한 필수적인 기능으로 볼 수 있다. 하지만 에이다는 큰 단점을 가지고 있었는데, 상속의 개념을 언어에 반영하지 않았다. 또한 대부분의 객체 지향 언어가 대부분 **동적 바인딩** 방식을 채택하고 있는 반면에 에이다는 **정적 바인딩**방식을 사용하고 있었다.

이후에는

객체 지향 언어는 프로그래밍 언어가 많은 지원을 받기 시작하고 발전하기 시작한 1990년 대 초반에 많은 발전이 있었다. 이들은 기본적으로 스몰토크와 에이다의 큰 틀을 따르고 있었지만 그들이 가지고 있었던 문제들을 해결해 나가는 과정으로 발전하였다. C++, 델파이, FoxPro와 같은 프로그램들은 객체 지향 언어에 가장 큰 영향을 미쳤던 GUI의 발전에 따라 점점 향상되었다.

그 중 AT&T의 벨 연구소에서 비야네 스트롭스트롭등에 의해 개발된 C++는 가장 많은 사용자를 확보하고 있는 객체지향 언어다. C++의 가장 큰 특징은 점진 진화해 가고 있다는 점이다. 초기의 C++는 C에 클래스 개념만 도입된 것에 지나지 않았으나, 중복, 상속, 가상 함수, 추상 클래스, 예외 처리와 같은 다양한 기능이 추가되면서 점차 향상되고 있다. C++는 C를 기반으로 하기 때문에 많은 프로그래머들의 인기를 받고 있지만 그로 인하여 객체 지향성을 제대로 반영하지 못하고 있다는 비난을 받기도 한다.

1990년대 중반 이후로 각광받고 있는 객체 지향 언어는 자바로 가진 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.

브래드 콕스가 개발한 오브젝티브-C는 C++와 마찬가지로 C와 객체 지향 언어를 혼합한 언어이다. 오브젝티브-C는 C++보다는 스몰토크에 좀 더 가깝게 정의된 언어이다.

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 defined data type)이라고 할 수 있다. 클래스는 다른 클래스 또는 외부 요소와 독립적으로 디자인하여야 한다. 프로그래머는 아니지만 해결해야 할 문제가 속하는 영역에 종사하는 사람이라면 클래스를 사용할 수 있다.
- 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.
- 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.

특징

객체 지향 프로그래밍의 특징은 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.

사람이 말로 표현 가능한 모든 것을 객체라 할 수 있다.

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 캡슐화한 것으로 접근 제어를 통해서 자료형의 정보를 은닉할 수 있다. 객체 지향 프로그래밍에서 일반적으로 추상 자료형을 클래스, 추상 자료형의 인스턴스를 객체, 추상 자료형에서 정의된 연산을 메소드(함수), 메소드의 호출을 생성자라고 한다.

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.

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. 자바는 지원하지 않는다.

다형성 개념

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 개수나 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, OOP의 경우 하나의 문제 해결을 위한 데이터를 클래스에 모아 놓은 데이터형을 사용함으로써 응집력을 강화하고, 클래스간에 독립적인 디자인을 함으로써 결합력을 약하게 한다.

객체 지향 언어

다음은 대표적인 객체 지향 프로그래밍 언어들이다.

- [시뮬라 67](#)([영어: Simula 67](#)): 최초의 객체 지향 언어로 알려져 있음
- [스몰토크](#)
- [비주얼 베이직 닷넷](#)
- [오브젝티브-C](#)
- [C++](#)
- [C#](#)
- [Dart](#)
- [자바](#)
- [객체지향 파스칼](#)
- [델파이](#)
- [파이썬](#)
- [펄](#)
- [루비](#)
- [액션스크립트](#)
- [ASP](#)
- [스위프트](#)

참고 문헌

- K교수의 객체지향 이야기, 김태균

같이 보기

- [구조적 프로그래밍](#)
- [CRC 카드](#)
- [NEXTSTEP](#)

외부 링크

- [\(영어\) 객체 지향 프로그래밍 \(https://curlie.org/Computers/Programming/Methodologies/Object-Oriented\)](https://curlie.org/Computers/Programming/Methodologies/Object-Oriented) - Curlie
-

원본 주소 "https://ko.wikipedia.org/w/index.php?title=객체_지향_프로그래밍&oldid=33784739"