# PYTHON PROGRAMMING

# OBJECT ORIENTED PROGRAMMING

# ABOUT OOP

# 객체 지향 언어



## ◆ 프로그래밍 패러다임

### ❖ 순차적 프로그래밍 (Sequential Programming)

- **명령어가 순서대로 실행되며 한 번에 하나의 작업만 처리**
- 시작점부터 끝까지 일련의 단계를 따라가며 실행
- **순차를 중점으로 보며 코드의 흐름과 순서에 기반해서 프로그래밍**
- 구조라는 개념이 없기 때문에 goto 문 사용
- 프로젝트 규모 커지고 더 복잡해질 수록, goto문 범람 ➔ **스파게티 코드**
- **중복 코드 존재**

# 객체 지향 언어



## ◆ 프로그래밍 패러다임

### ❖ 절차적 프로그래밍 (Procedural Programming)

- **중복 기능이나 관련 없는 기능 분리 및 구조 만든 프로그래밍 패러다임**
- 분리해낸 단위를 프로시저/ 함수
- 프로그램을 일련의 프로시저(절차 또는 함수)로 나누어 구성
  - → 각 프로시저는 특정 작업 수행
  - → 이러한 프로시저들이 순서대로 호출
  - → 전체 프로그램 수행
- 대표 언어 : C, Pascal, Fortran

# 객체 지향 언어



## ◆ 프로그래밍 패러다임

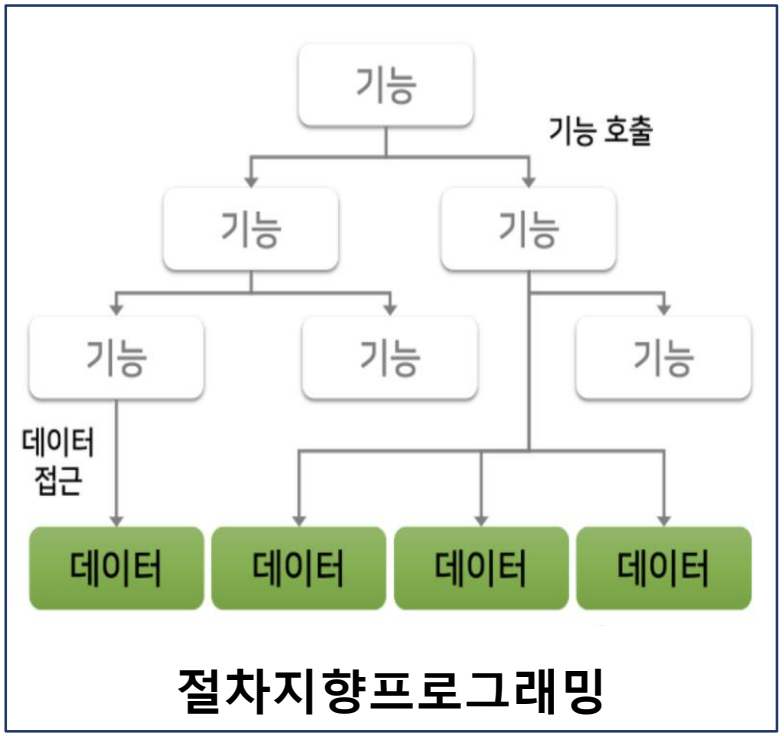
❖ 객체지향 프로그래밍 (Object Oriented Programming)

- 특정 개념의 **함수와 데이터 묶어서 관리하는 프로그래밍**
- 객체는 그 내부에 속성(Field)과 함수(Method) 존재
- 개발자들은 초반에 객체 설계할 때만 시간 소요
- 시간이 지날수록 중복 코드 최대로 줄일 수 있음
- **객체와 객체간에 독립성이 확립되므로 유지보수에 도움**
- 대표언어 : Java, C#, Objective-C, Python, ….

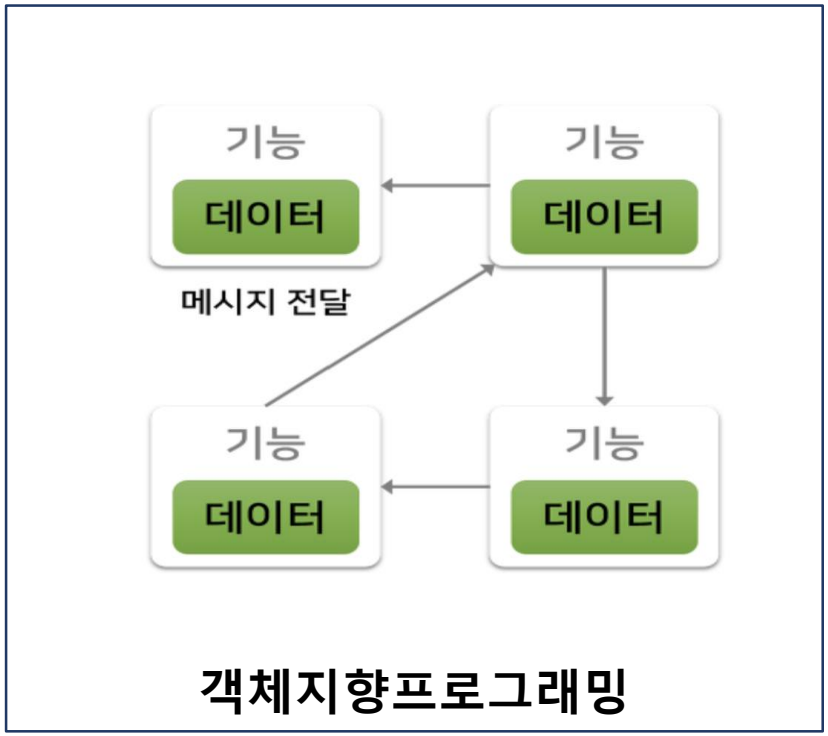
# 객체 지향 언어



## ◆ 프로그래밍 패러다임




절차지향프로그래밍




객체지향프로그래밍

# 객체 지향 언어



## ◆ 객체지향 프로그래밍 (OOP)

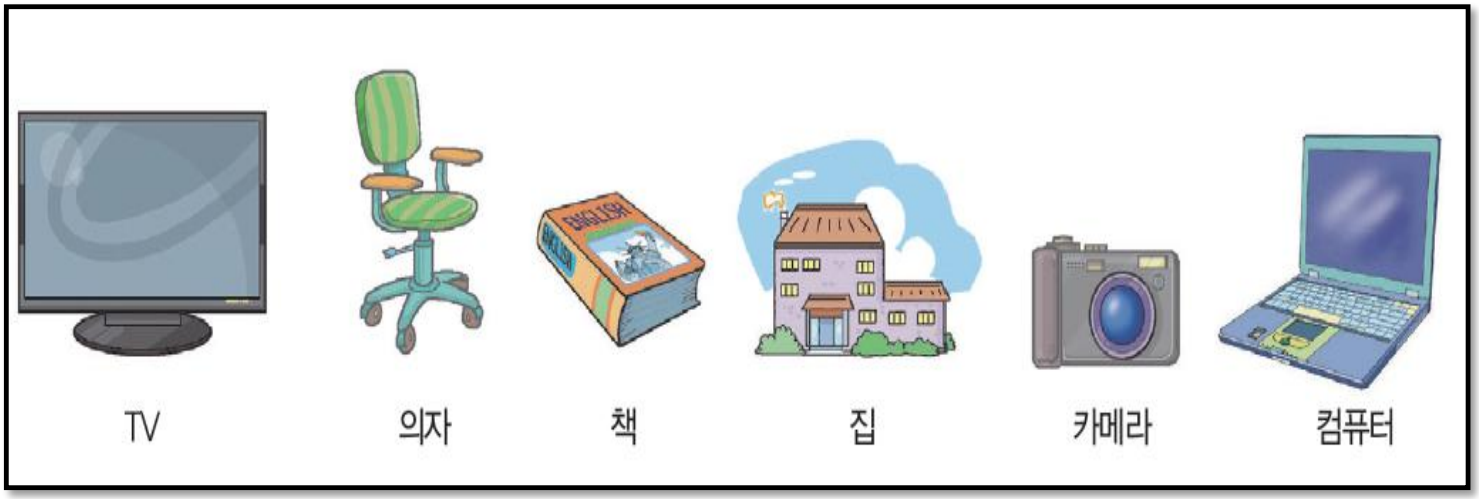
- **1960년 노위지안 컴퓨팅 센터의 조한 달과 크리스틴이 발표한 시뮬라 67**
- 시뮬라 67이 채택하고 있는 가장 중요한 개념은 클래스 도입
- 시뮬라 67의 발표 이후 10여년 간 객체 지향 언어는 전혀 주목 받지 못함
- **컴퓨터 프로그램을 명령어의 목록으로 보는 시각에서 벗어나 여러 개의 독립된 단위, 즉 "객체"들의 모임으로 파악하고자 하는 것**
- **소프트웨어 개발과 보수 간편, 보다 직관적인 코드 분석 가능**

# 객체 지향 언어



## ◆ 세상 모든 것이 객체(Object)

❖ 실세계 객체




- 객체마다 고유한 속성(attribute)과 행동(behavior) 가짐
- 다른 객체들과 정보를 주고 받는 등, 상호작용하면서 존재

# 객체 지향 언어 특성



## ◆ 추상화 (Abstraction)

- 사물의 **공통적인 특징(기능, 속성) 추출해서 정의**하는 것
- 실제로 존재하는 객체들의 공통적인 특성 파악한 후, 필요 없는 특성 제거해 하나의 묶음으로 만들어내는 과정 ➔공통점 묶기

- 객체지향적 관점에서 클래스 정의

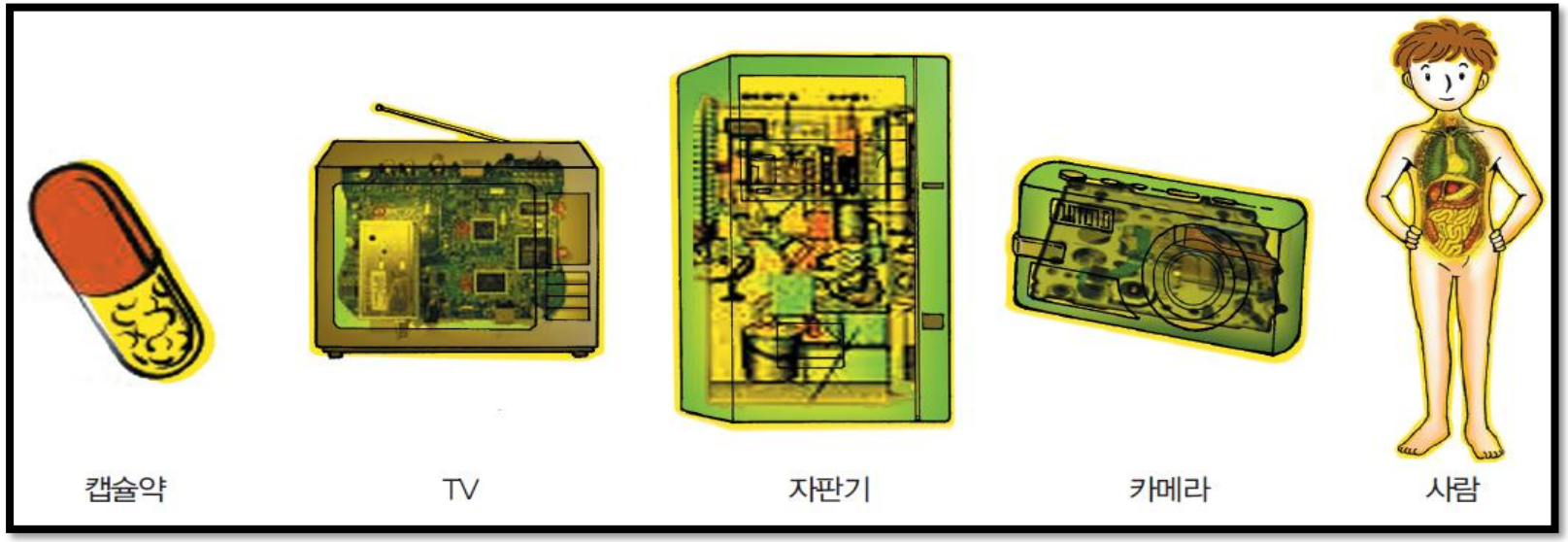
  → 공통된 성질만 추출하여 형상화/공식화 : 일반화(Generalization)

# 객체 지향 언어 특성



## ◆ 캡슐화 (Encapsulation)

- 객체를 캡슐로 싸서 내부/데이터를 볼 수 없게 하는 것
- 객체의 본질적인 특징 및 기능 ➔ 외부의 접근으로부터 객체 보호



# 객체 지향 언어 특성



## ◆ 캡슐화 (Encapsulation)

- **클래스(class)**
  - → 객체 모양 선언한 틀(캡슐)
  - → 메소드(멤버 함수)와 필드(멤버 변수)는 모두 클래스 내에 구현

- **객체(Object)**
  - → 클래스의 모양대로 생성된 실체(instance)
  - → 객체 내 데이터에 대한 보호, 외부 접근 제한
  - → 객체 외부에서는 비공개 멤버(필드, 메소드)에 직접 접근할 수 없음
  - → 객체 외부에서는 공개된 메소드를 통해 비공개 멤버 접근

# 객체 지향 언어 특성



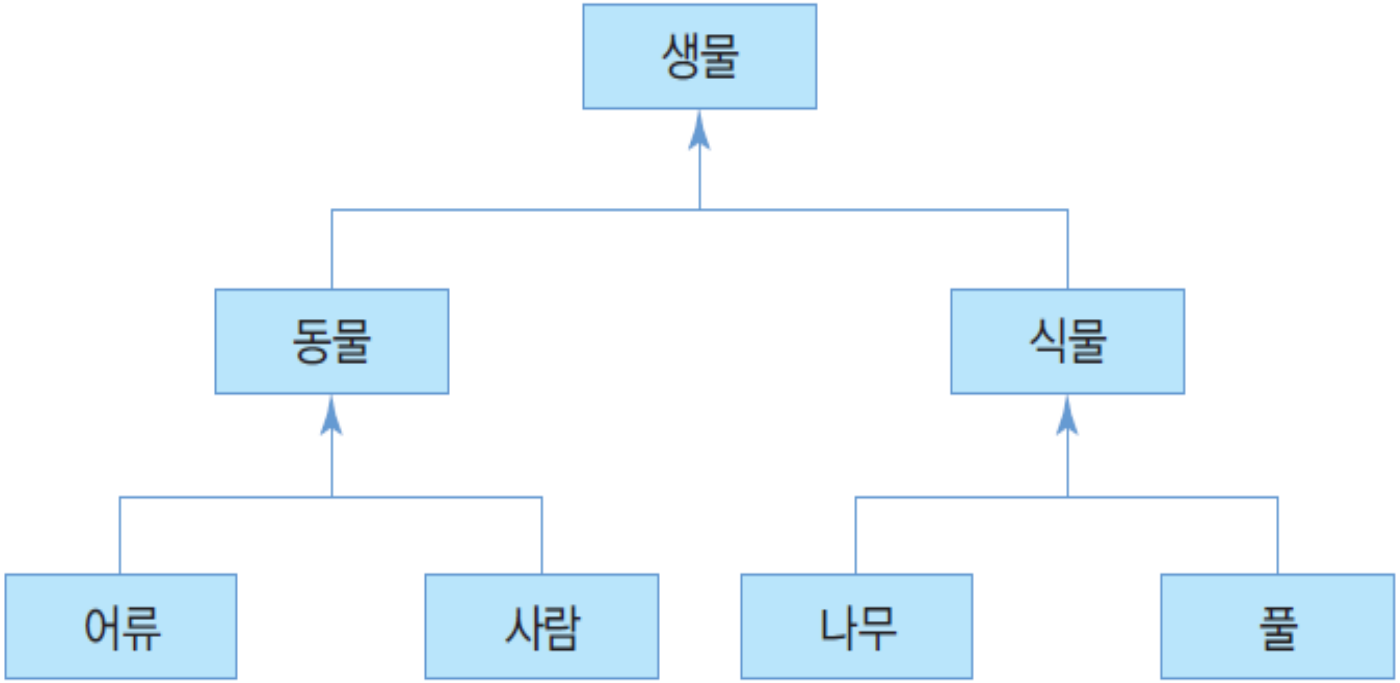
## ◆ 상속(Inheritance)

- 상위 개체의 속성이 하위 개체에 물려짐
- 하위 객체가 상위 개체의 속성을 모두 가지는 관계
- 코드의 재사용 및 확장
- 빠른 개발 진행

# 객체 지향 언어 특성



◆ 상속(Inheritance)



# 객체 지향 언어 특성



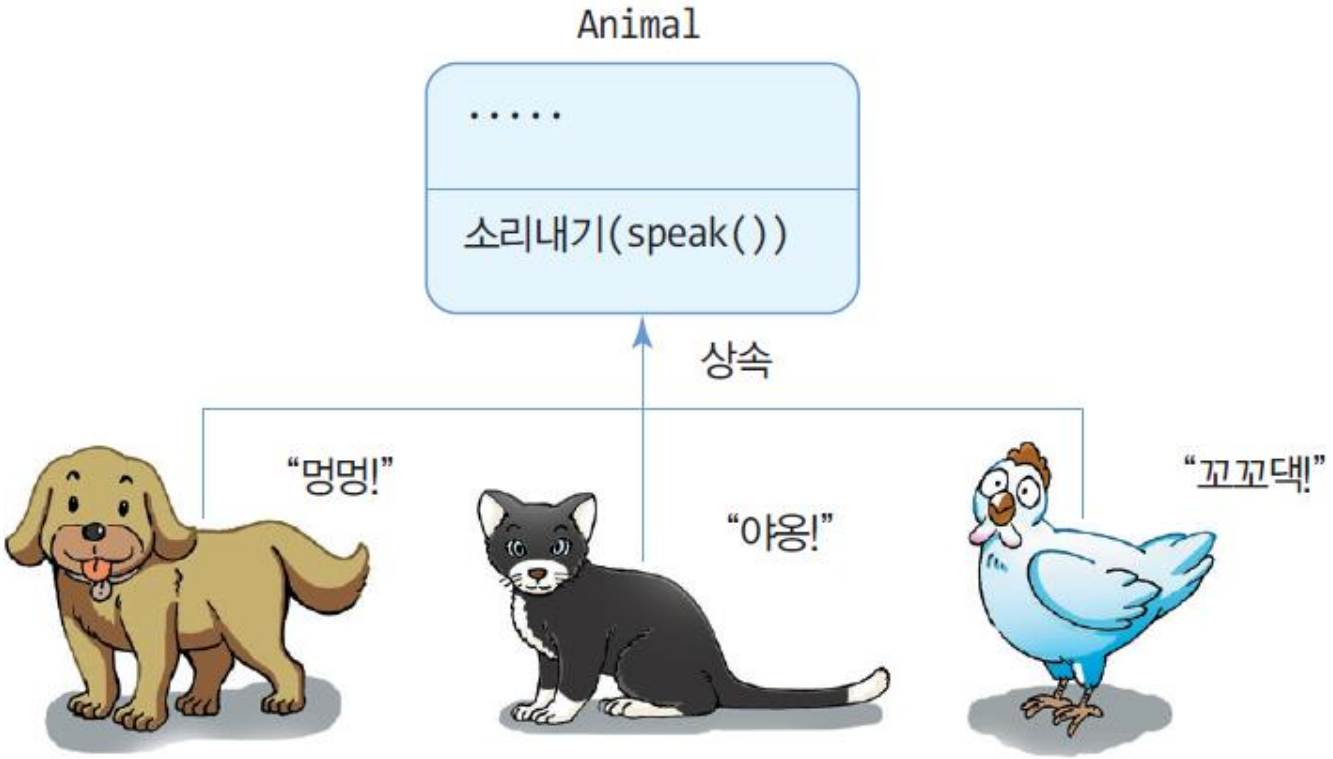
## ◆ 다형성(Polymorphism)

- 같은 이름의 메소드가 클래스나 객체에 따라 다르게 동작 구현
- 하나의 타입에 여러 객체를 대입할 수 있는 성질
- 부모 클래스 타입의 참조 변수로 자식 클래스 타입의 인스턴스를 참조할 수 있도록 하여 구현

# 객체 지향 언어 특성



◆ 다형성(Polymorphism)



# 객체 지향 언어 특성



## ◆ 장점

- 코드 재사용 용이
  모듈화된 객체, 그리고 상속을 통해 코드의 재사용 높일 수 있음
  .
- 생산성 향상
  생성된 클래스 상속 받거나, 객체 재사용, 부분 수정 등 높은 효율
- 자연적인 모델링 가능
  현실세계 개념 대입하여, 생각한 것 그대로 구현 가능
- 유지보수 용이
  프로그램 수정, 추가시 캡슐화로 유지보수 용이

# 객체 지향 언어 특성



## ◆ 단점

- 실행속도 느림

  컴퓨터 처리 구조 비슷한 절차지향 언어(C언어)보다 실행속도 느림

- 프로그램 용량이 커질 수 있음

  불필요한 정보들이 들어갈 수 있는데 프로그램의 용량이 증가될 수 있음

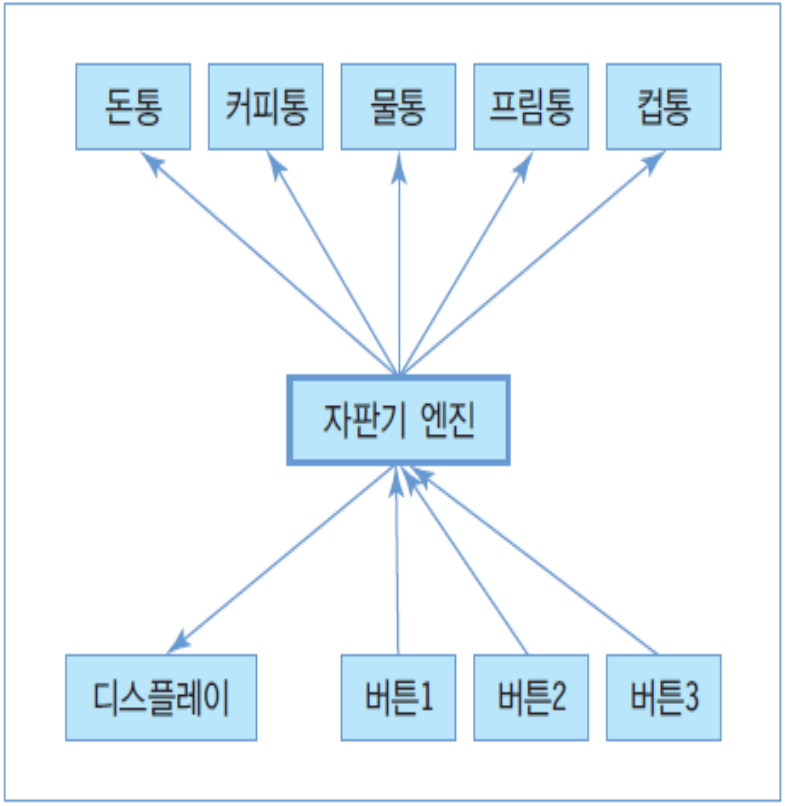
- 설계에 많은 시간 소요

  초기에 클래스별, 객체별, 상속 등의 구조 등을 모두 설계해야 하기 때문에 절차지향 언어에 비해 설계 시간이 많이 소요

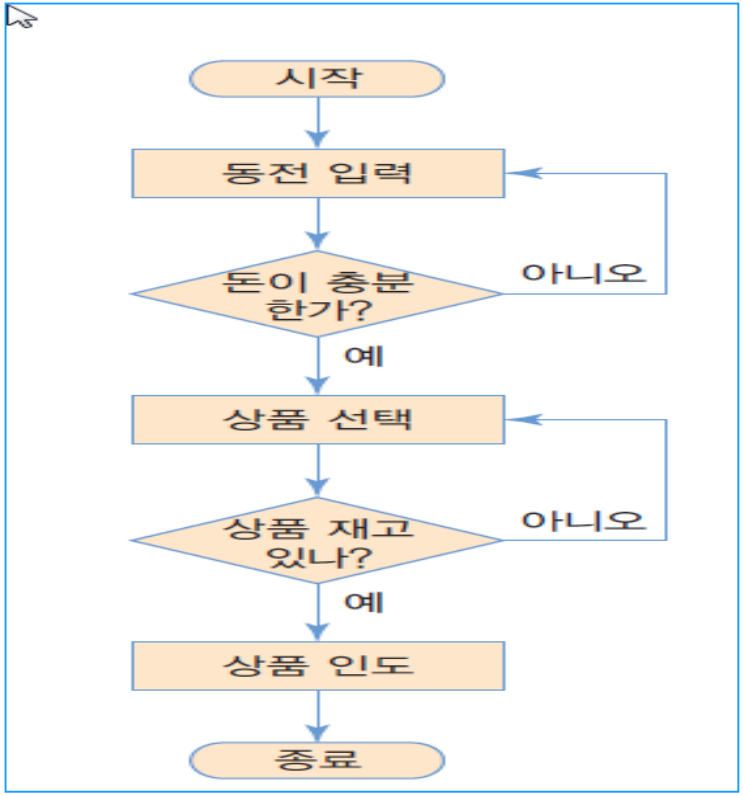
# 절차/객체 지향 프로그래밍



## ◆ 커피 자판기




객체지향적 프로그래밍의 객체들의 상호 관련성




절차지향적 프로그래밍의 실행 절차