Object Oriented 개념

경복대학교
소프트웨어융합과
배 희호 교수

Programmer의 자세

- 어떤 개발자가 될 것인가?
  - 개발자의 발전 단계
    - 단순 Coding(Coder) -> Programmer -> 분석/설계가(Analyst/Architect)
- 목적에 맞는 개발 Tool을 선정할 수 있는가?
  - 개발자는 새로운 개발 Tool에 항상 관심을 가져야 함
  - 각 개발 Tool에 대한 비교 Data를 만들어야 함
- Computer 세계의 흐름을 알고 있는가?
  - Computer 세계에 적응하고 흐름을 예측하기 위해서는 지나온 과거와 현재의 상태를 파악
  - Computer 관련 잡지를 일정 기간 꾸준히 구독
  - 용어와 약어에 대한 충분한 이해 필요
  - Hardware와 Software의 흐름을 동시에 파악

Programmer의 자세

- 객체 지향 개념을 알고 있는가?
  - 현존하는 개발 Tool을 좀더 체계적으로 배우기 위해서는 객체 지향 개념 공부 필요
- 문제를 해결하는 방법을 가지고 있는가?
  - 어려운 문제에 대한 해결 방법을 강구하기 위한 기존 자료의 활용, 전문가 자문, Internet Web Site 방문을 통해 방법 제시
  - 문제를 작은 부분으로 나누어 분석
  - 문제를 나누는 기준을 객체 지향의 객체화에 맞추어 분석
- “Know How”인가, “Know Who”인가, “Know Site”인가?
  - 과거에는 “Know How” 중시
  - 최근에는 “Know Who” or “Know Where”가 중시

Programmer의 자세

- 폭넓은 전문 분야 독서로 유관 분야 이해
  - Computer 이외의 관심 분야를 정해 그와 관련된 독서 필요
- 통신과 Database는 기본
  - 몇 년 전만해도 전문 분야였던 통신과 Database가 이제는 직접 개발을 담당하지 않더라도 개발 시 필수 고려 사항
  - JAVA는 통신과 Database 사용을 편리하게 제공
- 개발자는 예술가
  - 개발자는 개발 Tool과 Computer를 이용하여 상상력과 지식과 창조력으로 Code로 이루어진 Program을 작성
  - Program 개발자는 예술가(창조를 하는 직업)

Programming 방식의 변천

- 기계 중심의 Stored-Procedure Programming
  - 기계어, 어셈블리언어
- Structured Programming (Procedural-Oriented)
  - Pascal, C
  - 잘 정의된 제어 구조, Code Block, GOTO문 사용 억제, 순환 호출(recursion)과 지역 변수를 지원하는 독립형 SUB Program
  - 기능별 Module화
- Object-Oriented Programming
  - Smalltalk, C++, Java, Objective-C
  - 객체 단위로 Module화 (Data와 Control 통합)
  - 복잡도 감소 목표
  - Run-time시 정보 처리(dynamic binding)가 많아 Structure Programming 방식보다 실행 속도가 느림 (현대의 빠른 Computer로 극복)

좋은 Program 기준

- 1945, John Von Neumann, Stored Program
- 초기
  - 크기가 작은 Program
    - Memory 적게 차지해야 함
- 중기
  - 작고 빠른 Program
    - 실행 속도 중시

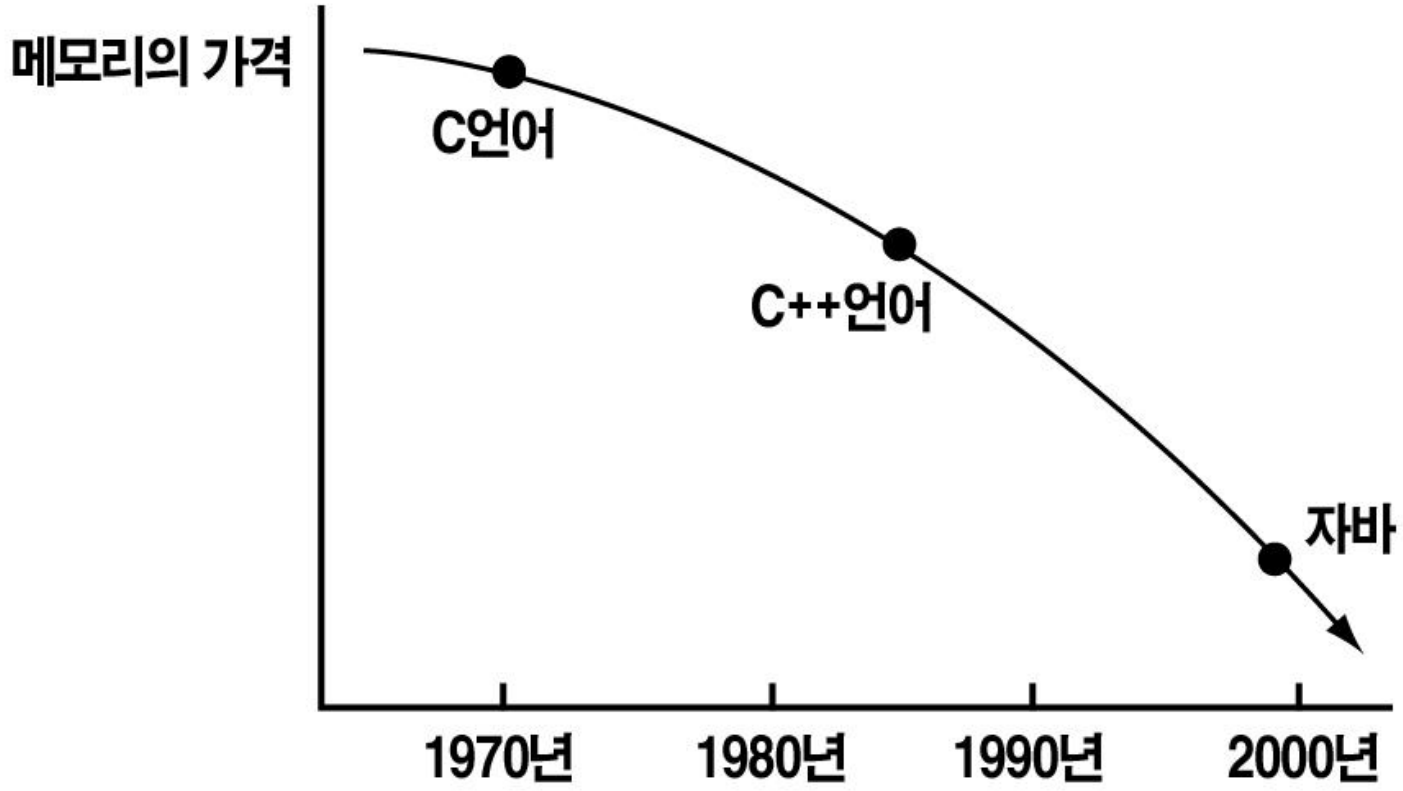
좋은 Program 기준

- 오늘 날
  - Memory 가격 ▼, 속도 ▲
  - 다수 인력에 의한 공동 개발
  - Program 크기 ➔ 수백만 Line
  - 거대한 Program 개발 요구(3R 기법)
    - 가독성(Readability) : 유지보수 시 얼마나 쉽게 이해
    - 신뢰성(Reliability) : 각각 개발부분에 대한 믿음
    - 재 사용성(Reusability) : 과거의 Code를 얼마나 재활용

Object Oriented Programming의 등장

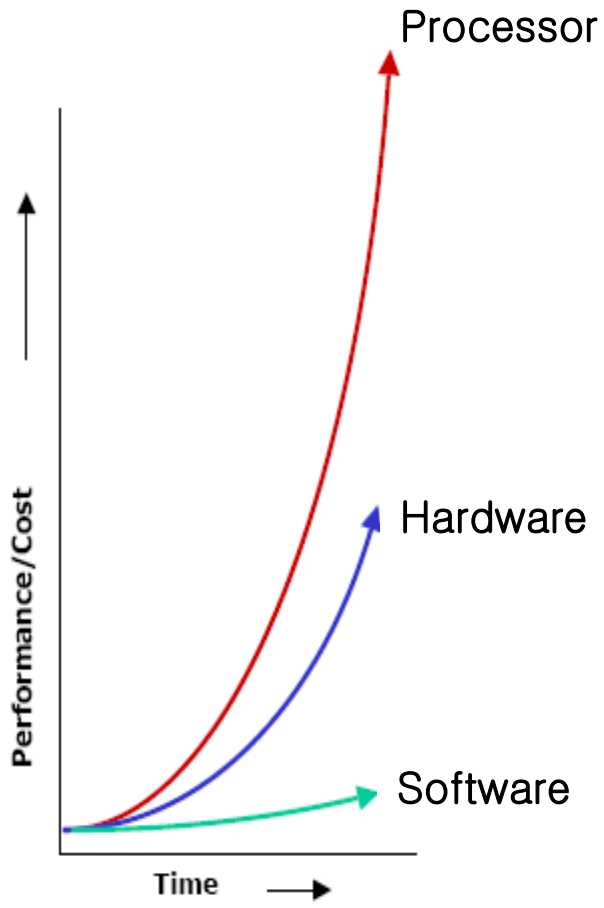
좋은 Program 기준

- Memory 가격과 Programming 기법



좋은 Program 기준

- Software의 Crisis
  - Software의 개발 생산성이 높아지지 못하였기 때문에 발생함
- Object Oriented Programming
  - Class와 Object를 기반으로 한 새로운 Programming 접근 방법
  - Object는 상태(state)와 행동(behavior)으로 구성됨



좋은 Program 기준

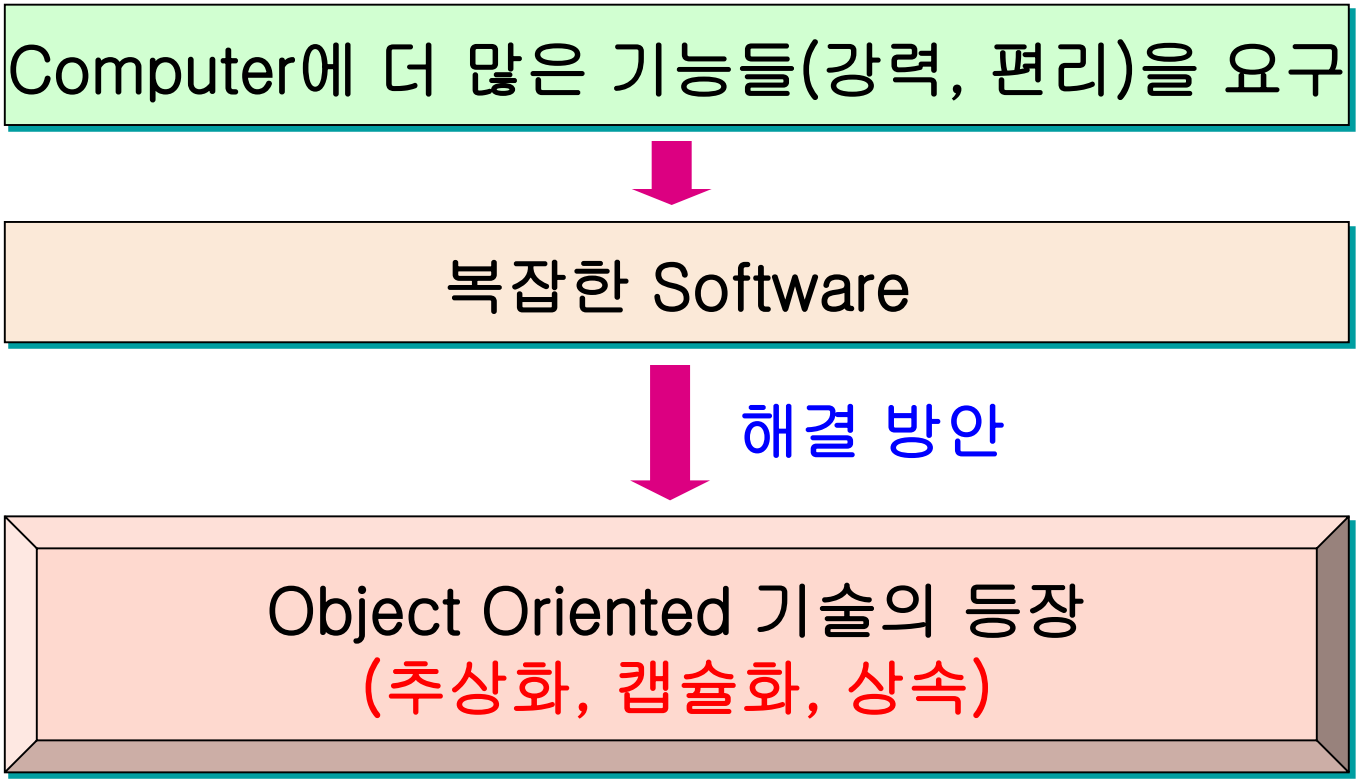
- Software의 생산성 향상
  - Computer 산업 발전에 따라 Software의 생명 주기(Life Cycle) 단축
  - Object Oriented Language는 Object(객체), Inheritance(상속), Polymorphism(다형성), Encapsulation(캡슐화) 등 Software 재사용을 위한 여러 장치 내장
    - Software의 재사용과 부분 수정을 통해 Software를 다시 만드는 부담을 대폭 줄임으로써 Software의 생산성이 향상

Programming 기법 진화

- 현실 세계에 대한 쉬운 Modeling
  - 과거
    - 수학 계산/통계 처리를 하는 등의 처리 과정, 계산 절차가 중요
  - 현재
    - Computer가 산업 전반에 활용
    - 현실 세계에서 발생하는 일을 Programming
      - 현실 세계에서는 절차나 과정보다 일과 관련된 물체(Object)들의 상호 작용으로 묘사하는 것이 용이
  - 현실 세계의 일을 보다 쉽게 Programming하기 위한 Object 중심의 Object Oriented Programming 탄생

Programming 기법 진화

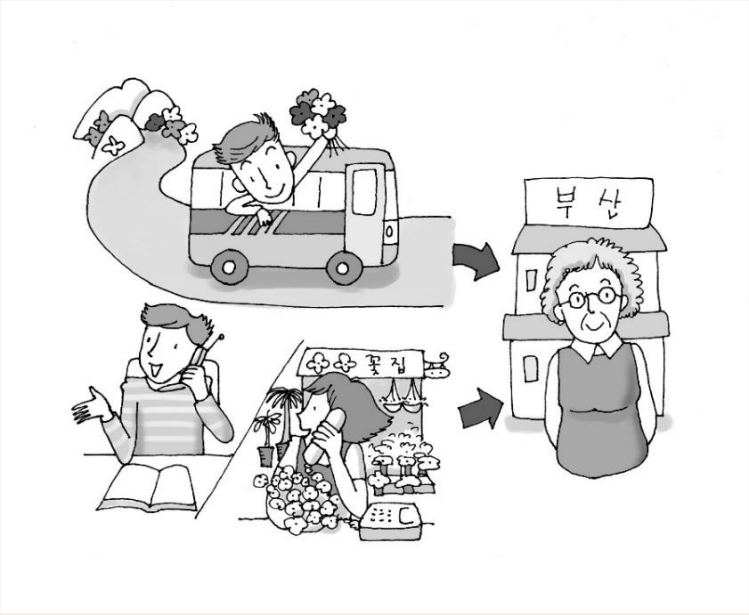
- Software의 복잡성



- 예) Windows Program : GUI 관련 일들 처리
  -> 문자 중심 Program보다 훨씬 복잡

Procedural-Oriented Programming

- 부산에 사시는 할머니에게 꽃을 보내자



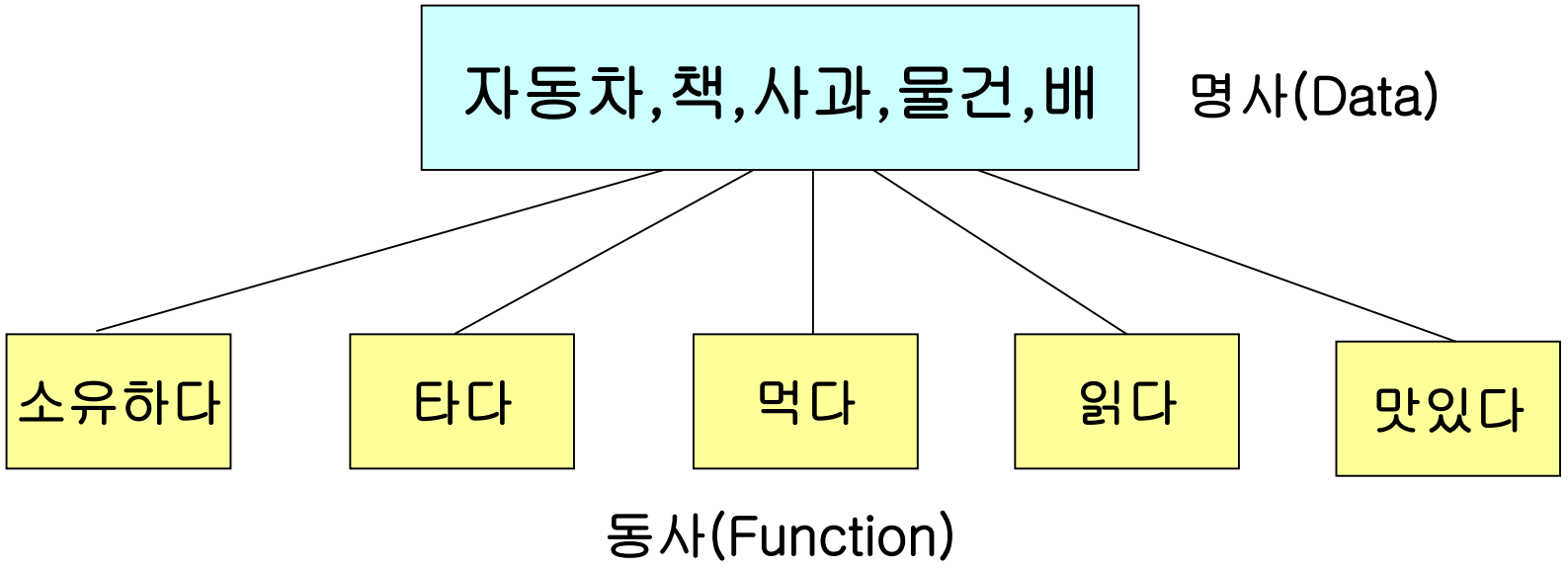
- Procedural Approach
  - 절차에 집중
  - 대부분의 Data가 공유 : 보호 안됨
  - 수정하기가 어려움
  - 관리하기가 복잡함

Procedural-Oriented Programming

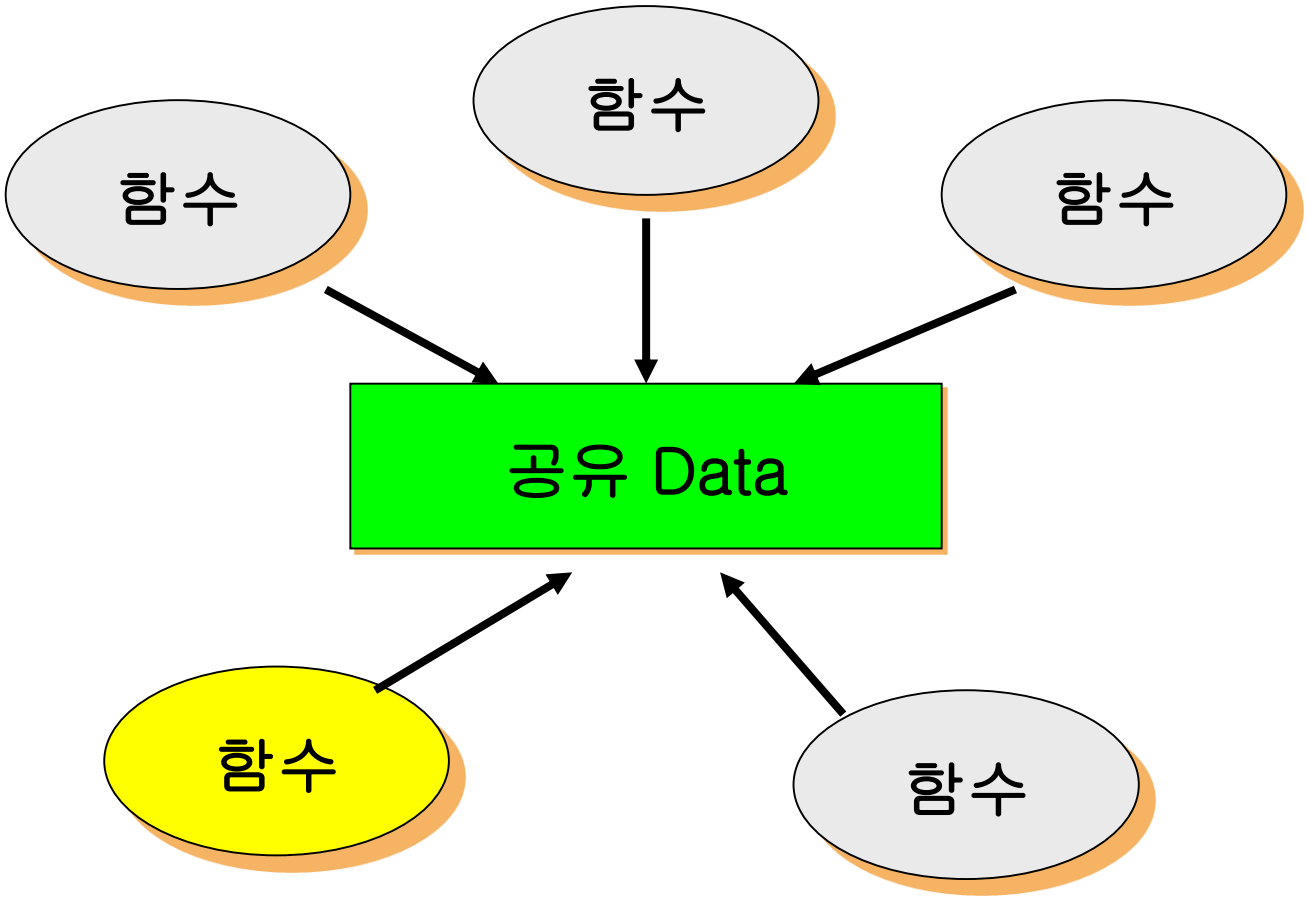
```
func1() { 꽃을 산다 }
func2() { 택배 직원을 부른다}
func3() { 택배 직원으로 하여금 꽃을 배송 시킨다}
func4() { 택배 직원은 부산까지 꽃을 배송한다}
func5() { 부산에 있는 택배 지국은 할머니에게 꽃을 보낸다}

main() {
    func1();
    func2();
    func3();
    func4();
    func5();
}
```

Procedural-Oriented Programming

- Software를 개발하는 과정은 먼저 사용자가 제공한 입력을 받고 이를 내부에서 정해진 일련의 처리 절차에 따라 가공한 후, 그 결과를 반환하는 Code를 작성하는 것임
- 예) C, Pascal, Fortran,…



Procedural-Oriented Programming



Procedural-Oriented Programming

- Procedural-Oriented Language에서의 Program들의 대부분은 Data를 변화시키는 Algorithm으로 구성되며, Data와 그 Data를 변화시키는 Algorithm이 분리된 형태를 취함
- Program을 작성하기 위해서는 모든 Data 구조를 이해하고 있어야 함
- Program이 커지거나 복잡해지면 Program이 혼란스럽게 되어 Error를 찾는 Debugging 및 Program의 Maintenance가 어려워 짐

Procedural-Oriented Programming

- 문제 – '해야 할 일의 순서'
- Data를 Record, Structure(구조체)로 묶음
- Data 조작을 위한 Function(함수), Procedure 작성
- 이 Function은 Data와 별개로 해야할 순서에 따라 호출, 해야할 일에 따라 Data 조작
- 초점 – 일의 처리 순서, Data 조작 함수
- 문제점
  - 예) 전역 변수
    - 값 손상 위험
    - Data의 순서 변경 시 관련 함수를 모두 수정해야 함

Structured Programming

- 모든 Program은 순차, 선택, 반복으로 구성할 수 있음
  - Concatenation(순차)
    - 구문 순서에 따라서 순서대로 수행된다는 것
  - Selection(선택)
    - Program의 상태에 따라서 여러 구문들 중에서 하나를 수행하는 것
    - 예) if..then..else..endif, switch..case와 같은 Keyword로 표현
  - Repetition(반복)
    - Program이 특정 상태에 도달할 때까지 구문을 반복하여 수행하거나, 집합체의 각각의 원소들에 대해 어떤 구문을 반복 수행하는 것
    - 예) while, repeat, for, do..until 같은 Keyword로 표현

Structured Programming

- 종종 반복 영역의 시작점을 하나로 하는 것이 추천되며, (원조 Structured Programming에서는 종료점도 하나로 해야한다고 추천하고,) 몇 가지 언어에서는 이것을 꼭 지켜야 하도록 하고 있음
- 잘 정의된 제어 구조(Well-Designed Control Structure), Code block, GOTO문장 사용 억제, 재귀 호출(Recursive Function : 순환 호출)과 지역변수를 지원해주는 독립형 Program 등에 기반을 둔 Programming
- Program의 Line수가 늘어나게 되면 관리/수정이 어려움
  - 어느 정도의 복잡성을 가진 Program에서는 아주 좋은 결과를 볼 수 있으나 특정 한계를 넘어서는 여러 면에서 문제가 발생할 수 있음

Procedural-Oriented Programming

- Procedural-Oriented 방식은 “Software의 Crisis” 라는 문제를 발생
  - Software의 복잡성 증가
  - 개발과 유지 및 보수에 많은 비용 발생

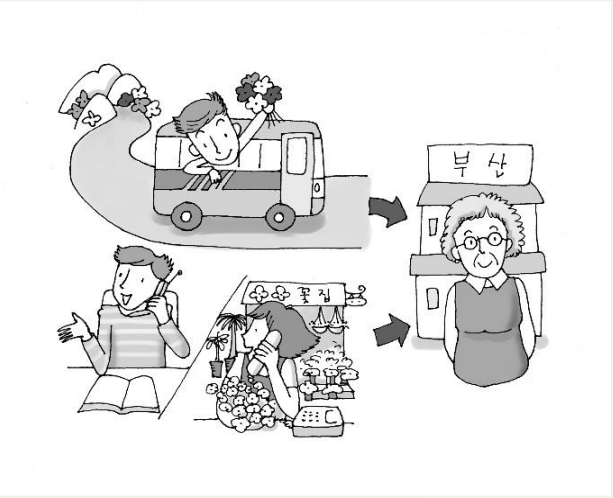
- 이러한 Procedural-Oriented 방식의 문제점으로 인해 모든 것은 Object의 상호작용을 통해 이루어 진다는 Object Oriented 개념을 도입

객체지향 프로그래밍의 등장

Object Oriented Programming

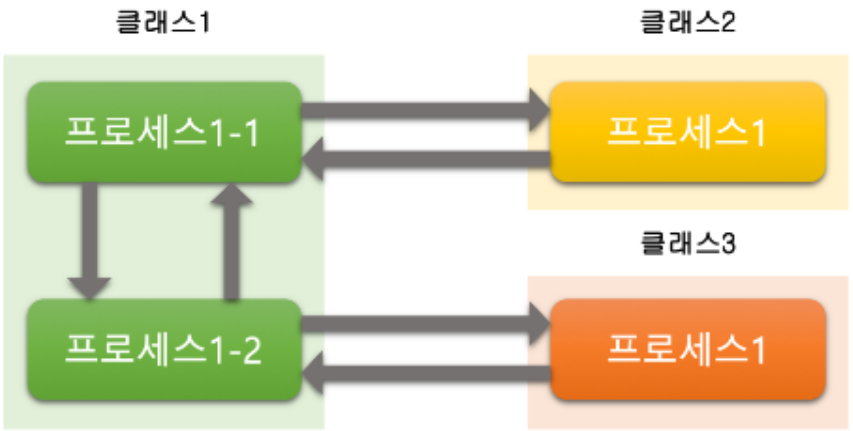
- 부산에 사시는 할머니에게 꽃을 보냄

```
class 나자신 { … }
class 꽃판매원 { … }
class 택배직원 { … }
class 택백부산지국 { …}
class 할머니 {… }
main( ) {
    나자신 a; 꽃판매원 b; 택배직원 c;
    택배부산지국 d; 할머니 e;
    a.buyFlower(b);
    a.call(c);
    c.deliverFlowerTo(d);
    d.deliverFlowerTo(e);
}
```



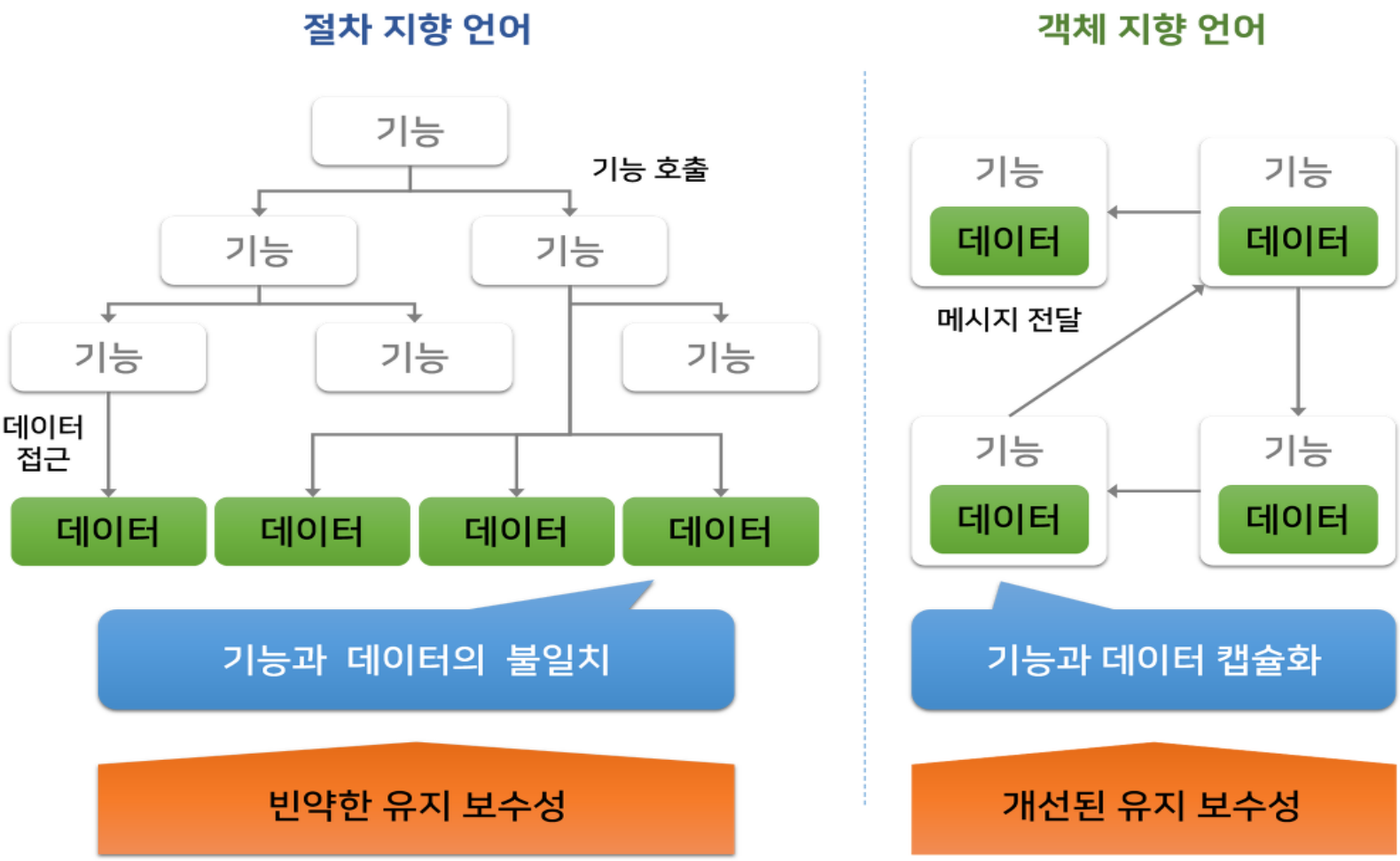
절차 지향과 객체 지향

절차지향 프로그래밍



객체지향 프로그래밍

절차 지향과 객체 지향


절차 지향 언어
기능
기능 호출
기능
기능
기능
기능
기능
데이터 접근
데이터
데이터
데이터
데이터
기능과 데이터의 불일치
빈약한 유지 보수성
객체 지향 언어
기능
데이터
기능
데이터
메시지 전달
기능
데이터
기능
데이터
기능과 데이터 캡슐화
개선된 유지 보수성

절차 지향과 객체 지향

- 목적
  - 절차지향 언어를 사용한다면, 말 그대로 실행 순서, 즉 절차가 더 중점이 됨
  - 객체지향 언어를 사용한다면, 필요한 객체들의 종류와 속성 등이 더 중점이 됨

- 설계 방식
  - Procedural-Oriented Programming은 Program의 순서와 흐름을 먼저 세우고 필요한 자료 구조와 함수들을 설계하는 방식
  - Object-Oriented Programming은 반대로 자료 구조와 이를 중심으로 한 Module들을 먼저 설계한 다음에 이들의 실행 순서와 흐름을 조합하는 방식

절차 지향과 객체 지향

	장점	단점
절차적 프로그래밍	✓ 컴퓨터 처리 구조와 유사하여 실행 속도가 빠름	✓ 실행 순서가 정해져 있어 Code 순서가 바뀌면 동일한 결과를 보장 못함 ✓ Debugging이 어려움 ✓ 유지보수가 어려움
객체 지향 프로그래밍	✓ Program 신뢰성이 높아짐 ✓ Code 재사용이 쉬워짐 ✓ Upgrade 쉬움 ✓ Debugging이 쉬움	✓ 처리 속도가 절차적 프로그래밍보다 느림 ✓ 설계에 많은 시간이 듦

절차 지향과 객체 지향

- 객체 지향 방법론의 이점
  - 실 생활의 개념과 거의 동일함
  - Programmer가 작성해야 할 Program의 양이 줄어 듬
    - 각 객체들마다 완벽한 모듈화(Modularity)를 제공하기 때문에 코드 재사용(Code Reuse)을 극대화 할 수 있음
  - 유지보수가 편리함

Object Oriented 개념

- 다음 그림을 보고 의자인 것은 어느 것인가?

A

B

C

- 선택한 그림이 의자라고 생각하는 이유는?

Object Oriented 개념

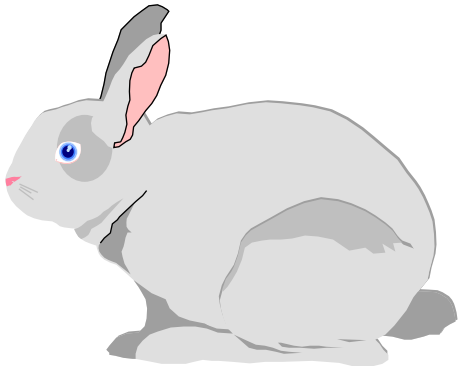
- 현실 세계의 물체를 Object(객체)로 사상할 때
  - 현실의 모든 측면을 Object(객체)로 표현하지는 않고
  - 문제의 중요한 측면, 주목하고 싶은 측면을 강조하여 표현
- 복잡한 문제를 다루는 가장 기본이 되는 메커니즘
- 문제에서 반드시 필요한 근본적인 특성에 집중하고, 중요하지 않는 부분이나 상세 내역들을 제거시켜 나가는 과정
- 예) '철수'와 '영희'등 회사원들이 A회사에 근무
  - 철수, 영희 : 현실 세계에 존재하는 개개의 객체
  - 회사원 : 회사에 다니는 사람들의 공통적인 특성을 추상적으로 표현

Object Oriented 개념

- 추상화(Abstraction) 과정에는 반드시 관점(view point)이 작용
  - 주어진 관점에 따라 추상화 결과는 달라짐
  - Software 개발에서 관점이란 주어진 도메인이 됨

- 아래 그림에서 각 관점에 따른 토끼의 속성과 행위는?

해부학자



미식가

Object Oriented 개념

- 사물을 인식할 때 사람은 자신이 알고 있는 사물의 특징을 떠올림
  - 사물의 모양, 사물의 기능 등...
  - 정확한 인식을 위해 사전에 정의된 내용을 습득
- 새로운 사물을 인식하려고 할 때 이미 알고 있는 지식을 기반으로 함
  - 예) 눈을 전혀 보지 못한 나라에 사는 사람들은 옆의 그림을 어떻게 인식할 것인가?

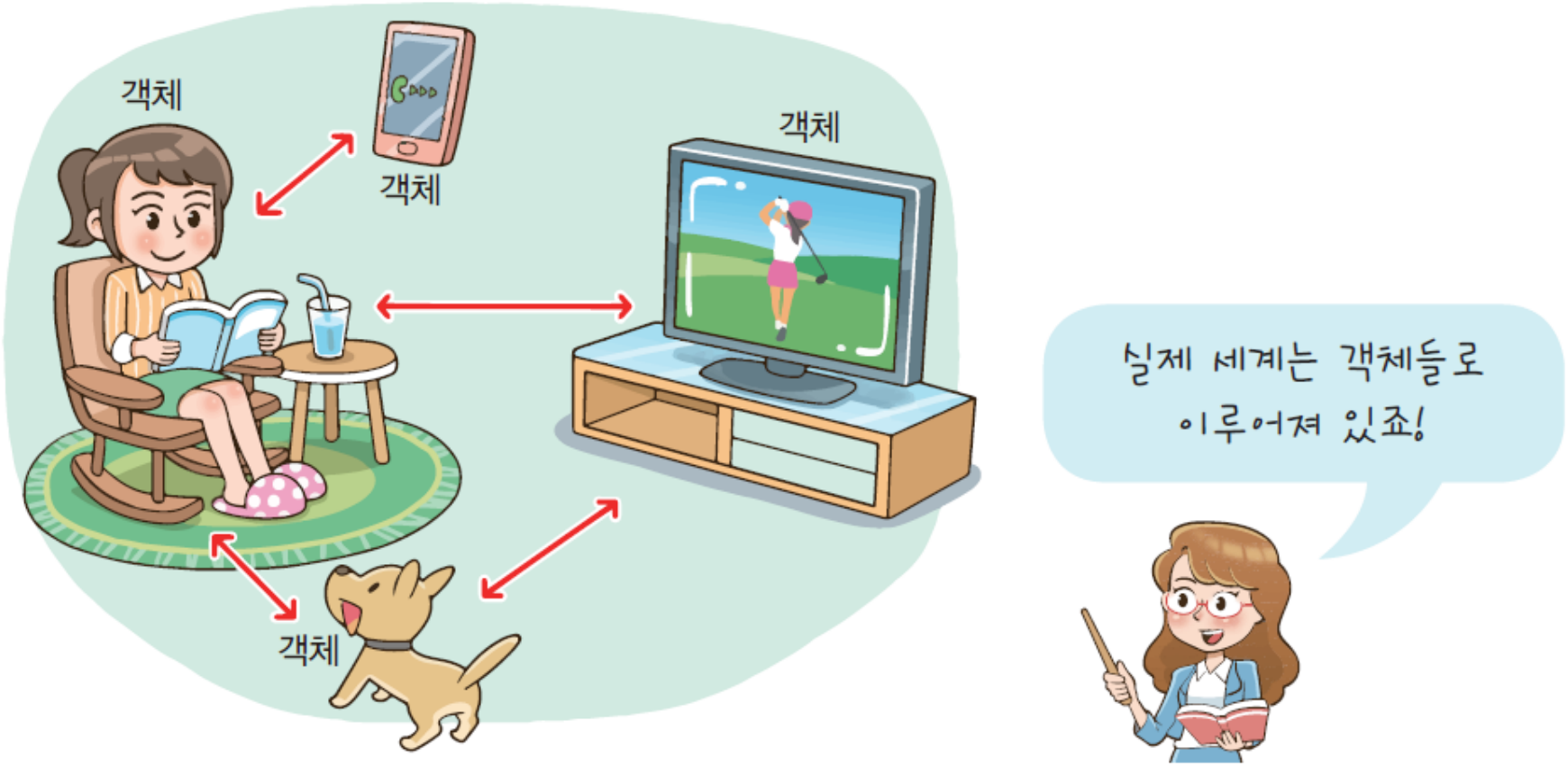
이와 같이 사람이 어떤 대상(사물)을 인식하고 처리하는 과정을 그대로 반영하기 위한 것이 Object Oriented

Object Oriented 개념

- Object Oriented Language에서의 Program들은 객체(Object)들의 집합
- Program은 이러한 Object들을 생성하고 Object들간에 메시지(Message)를 통하여 정보를 교환함으로서 Programming이 이루어짐
- 각각의 Object들은 자신이 가지는 고유의 Data와 그 Data를 처리할 수 있는 절차인 Method(메소드)를 가지고 있음
- Object Oriented Language에서는 Data와 그 Data를 처리할 수 있는 Method를 하나의 단위인 객체(Object)로 표현

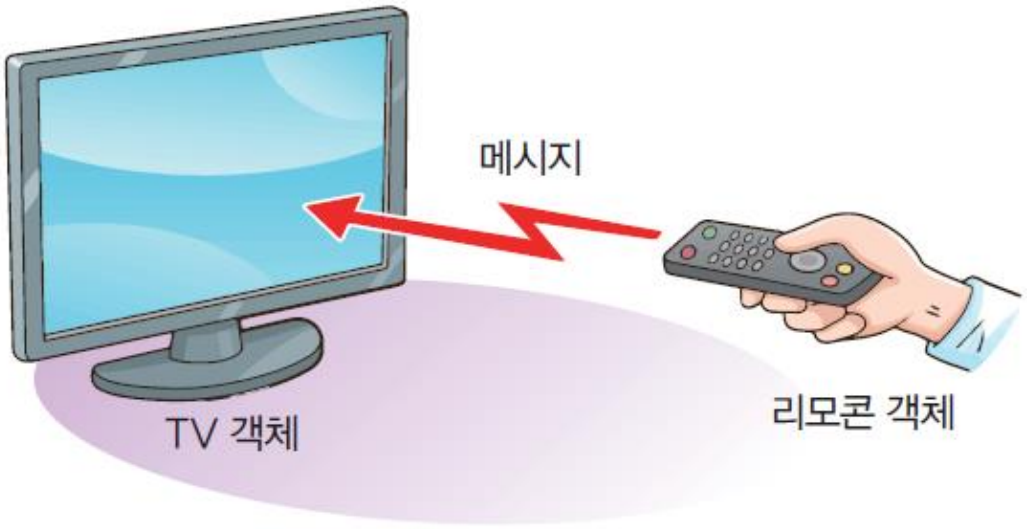
Object Oriented 개념

- 실 세계(Real World)는 객체(Object)로 이루어짐



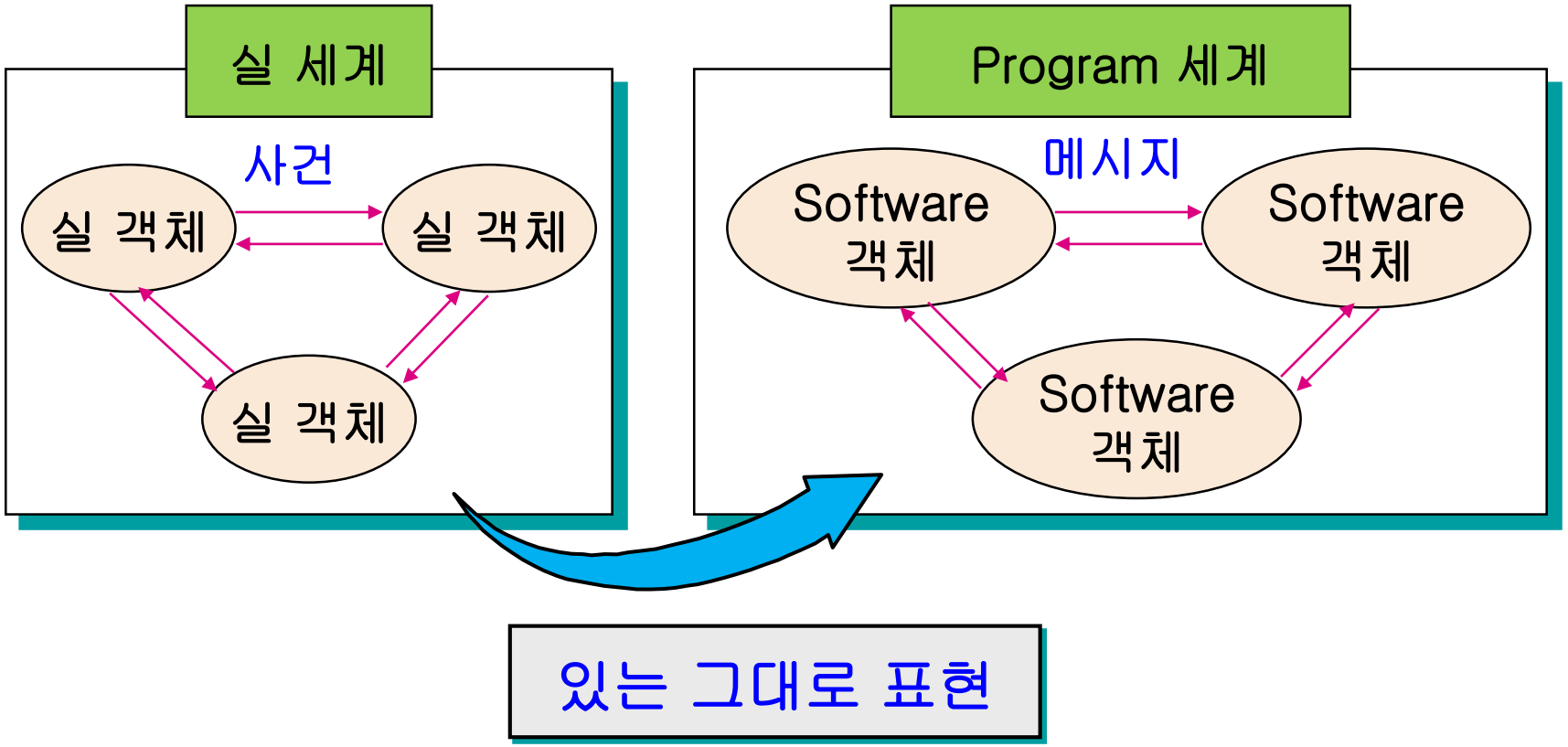
Object Oriented 개념

- Object와 Message



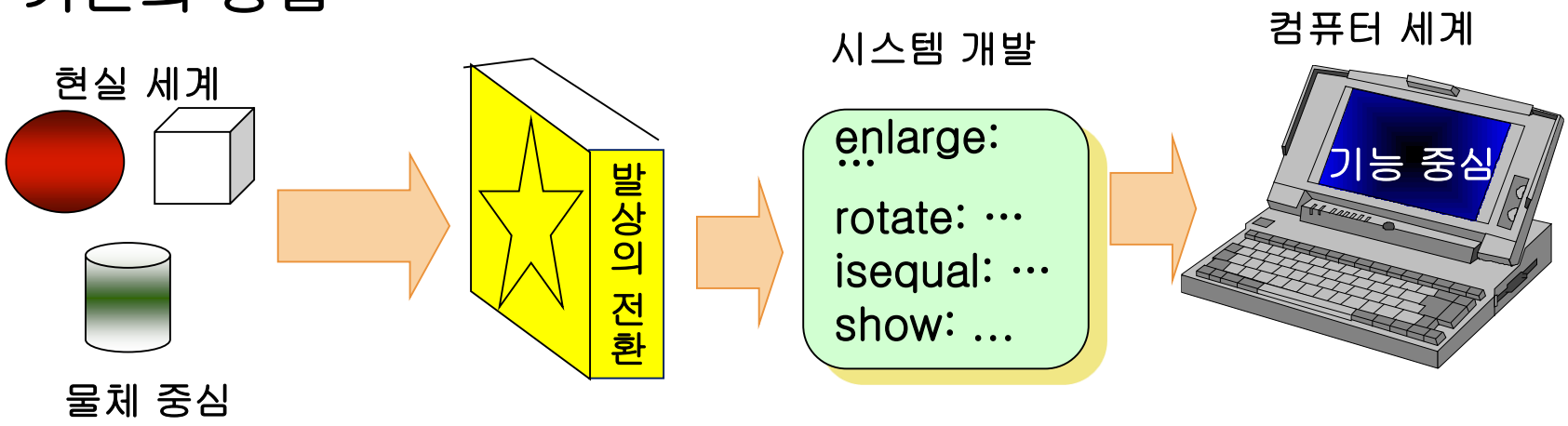
Object-Oriented 개념

- 실 객체를 표현한 Software 객체로 Program 구성
- Software 객체들의 상호 동작으로 Program 수행

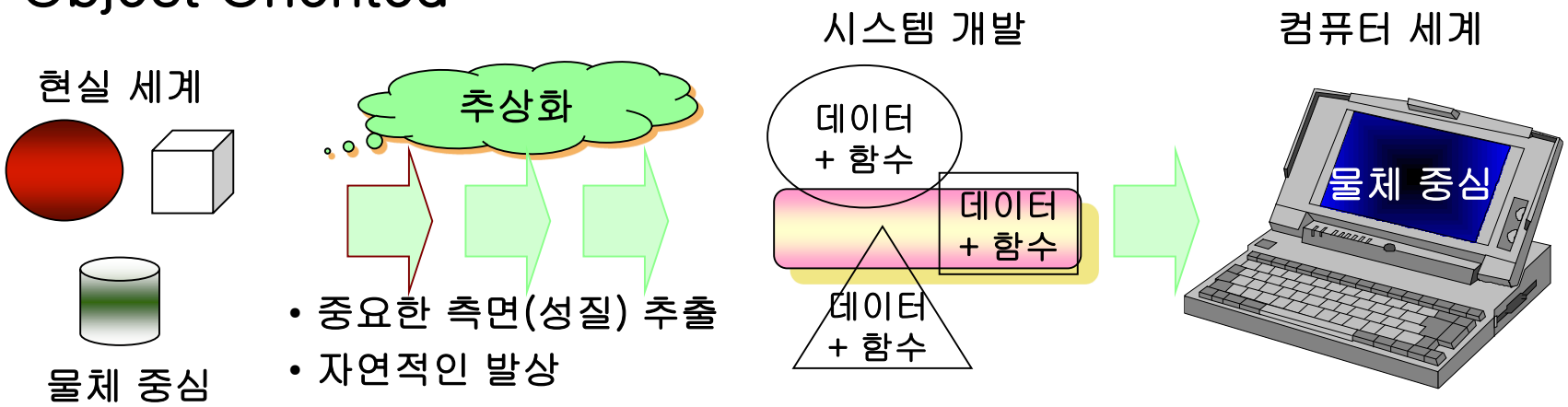


Object Oriented 개념

기존의 방법

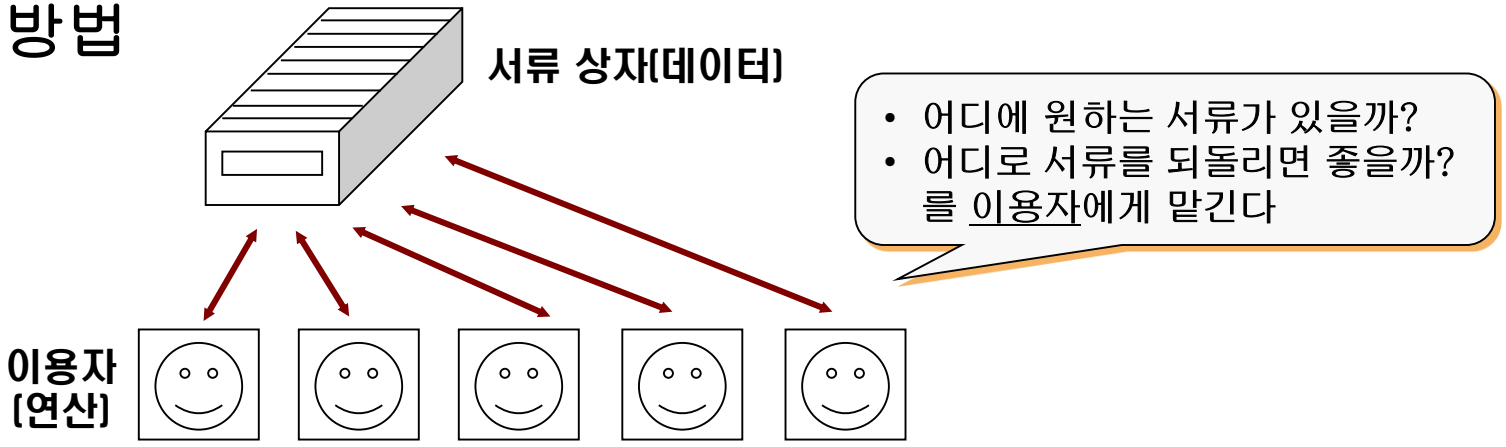


Object Oriented

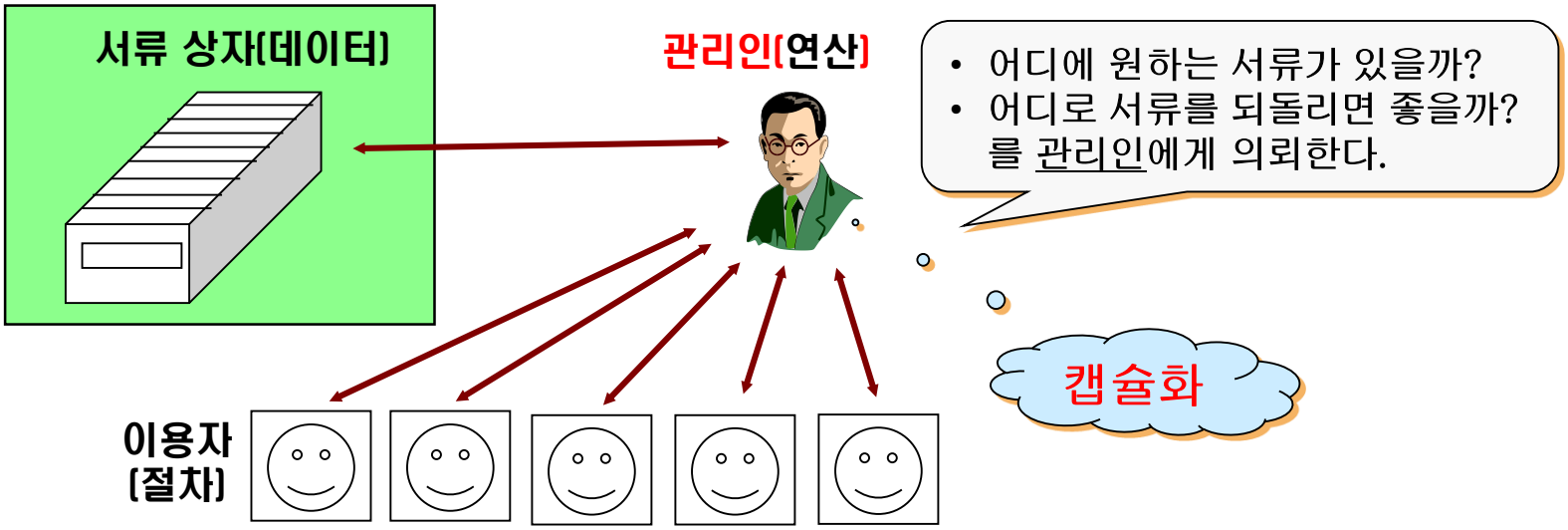


Object Oriented 개념

- 기존의 방법

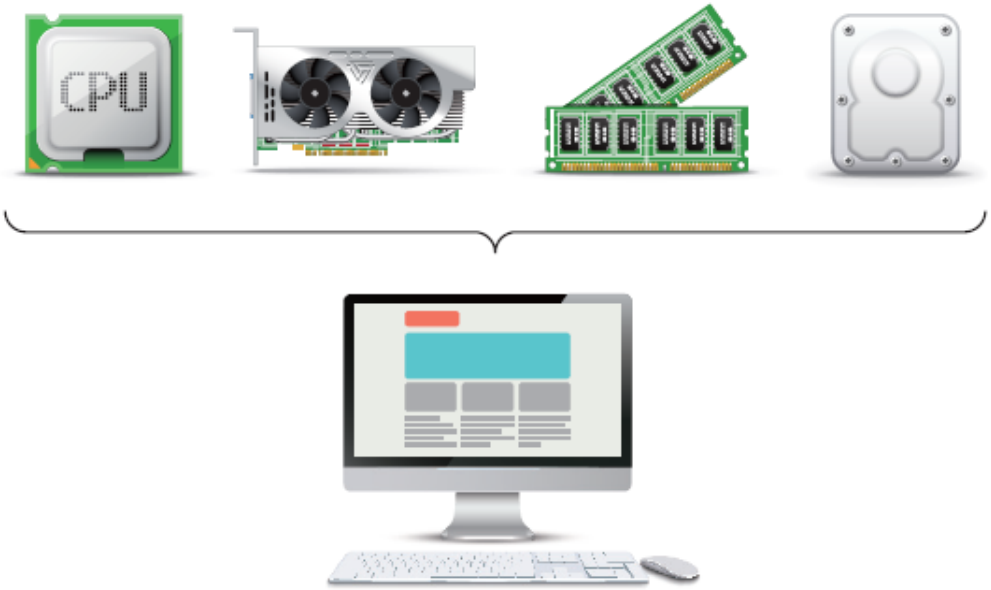


- Object Oriented

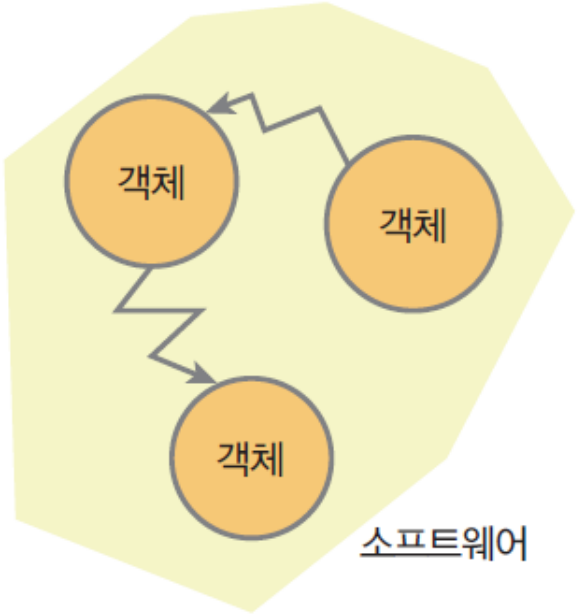


Object Oriented 개념

- Object Oriented로 Software를 작성하는 것은 Computer 부품을 구입하여서 Computer를 조립하는 것과 비슷함

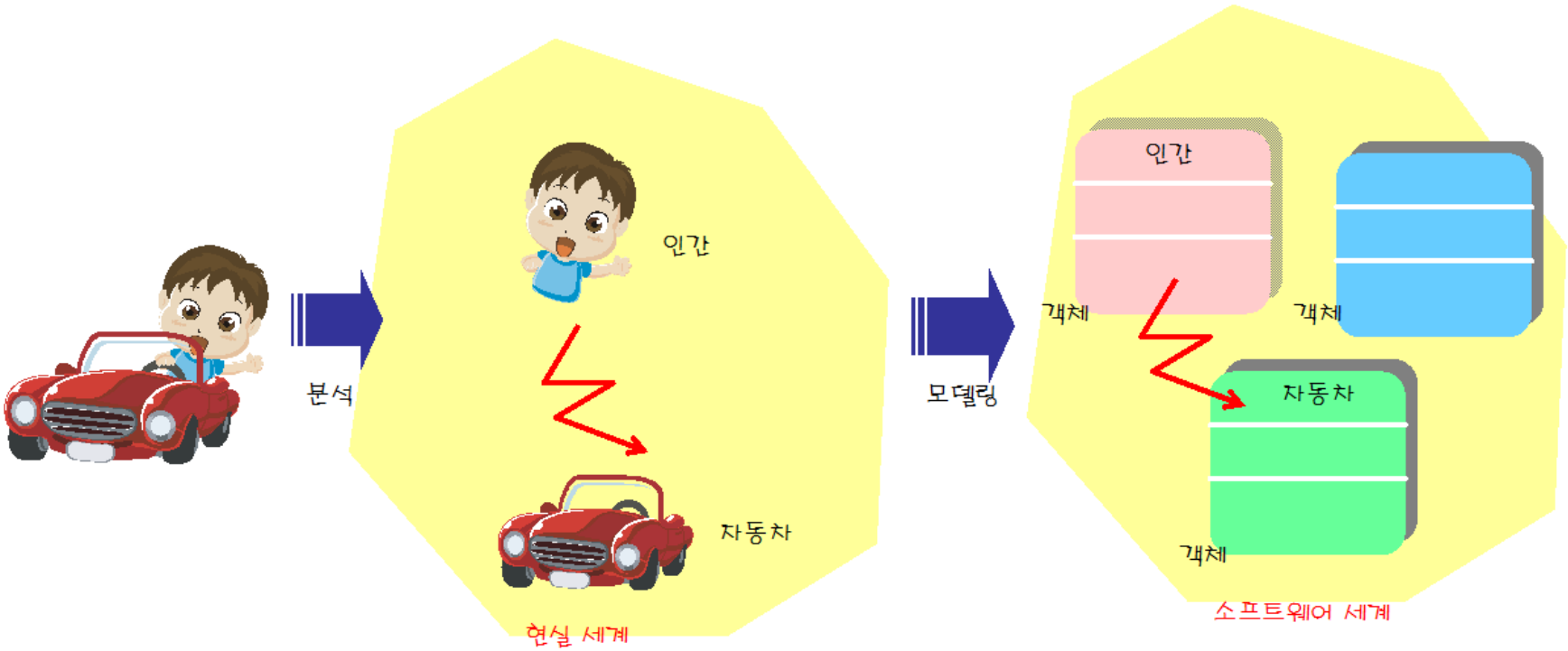


부품을 조립하여 제품을 만들듯이
객체를 조합하여 소프트웨어를 만든다



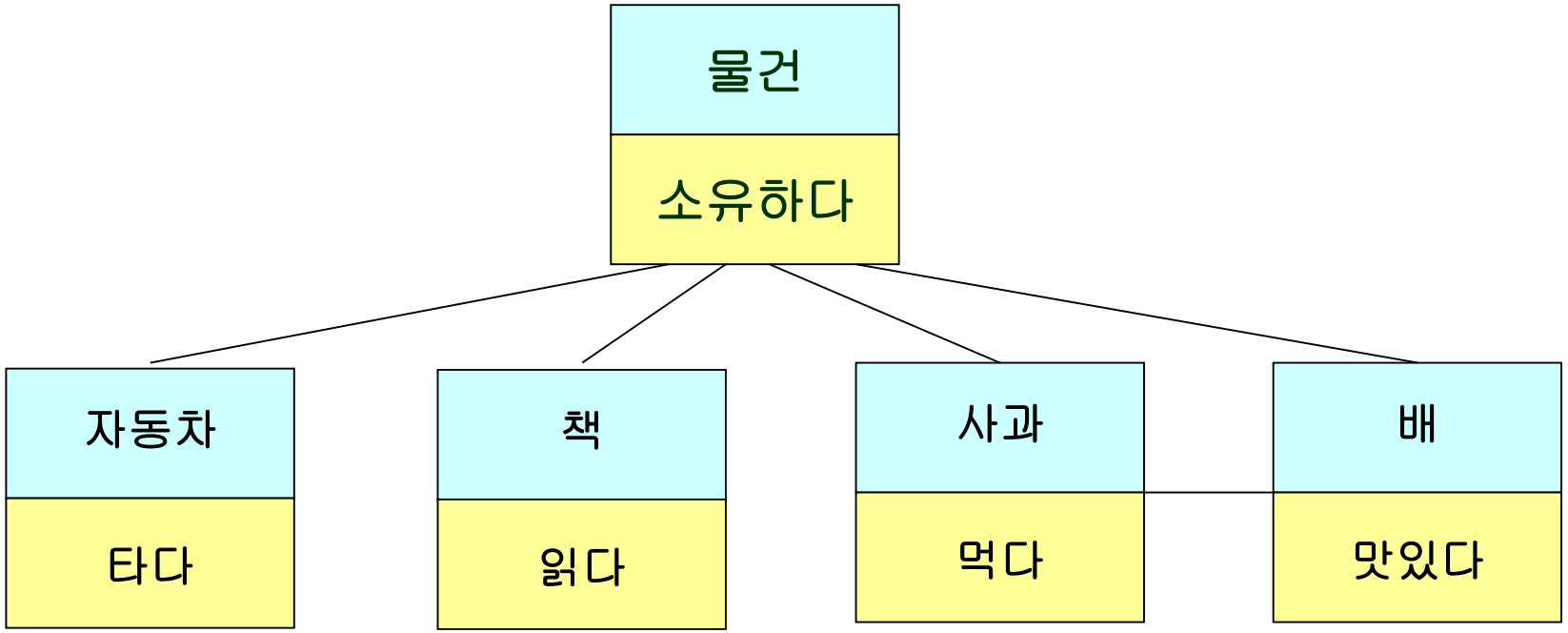
Object Oriented 개념

- 실 세계를 Modeling하여 Software를 개발하는 방법



Object Oriented 개념

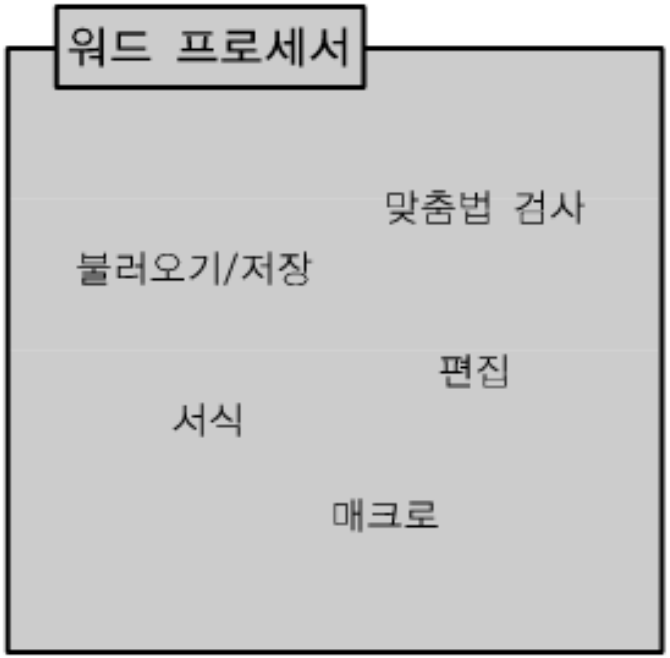
- 물건 Object는 다른 객체 보다 상위 관계를 갖는 Object이고 이 Object에서 갖는 성질은 다른 Object도 공유
- 비슷한 성질의 사과, 배는 서로 성질을 공유하는 관계



Object Oriented 개념

- Object Programming은 부품을 모아서 조립하는 과정에 비유할 수 있음
- 여기서 객체(Object)를 부품이라고 볼 수 있음

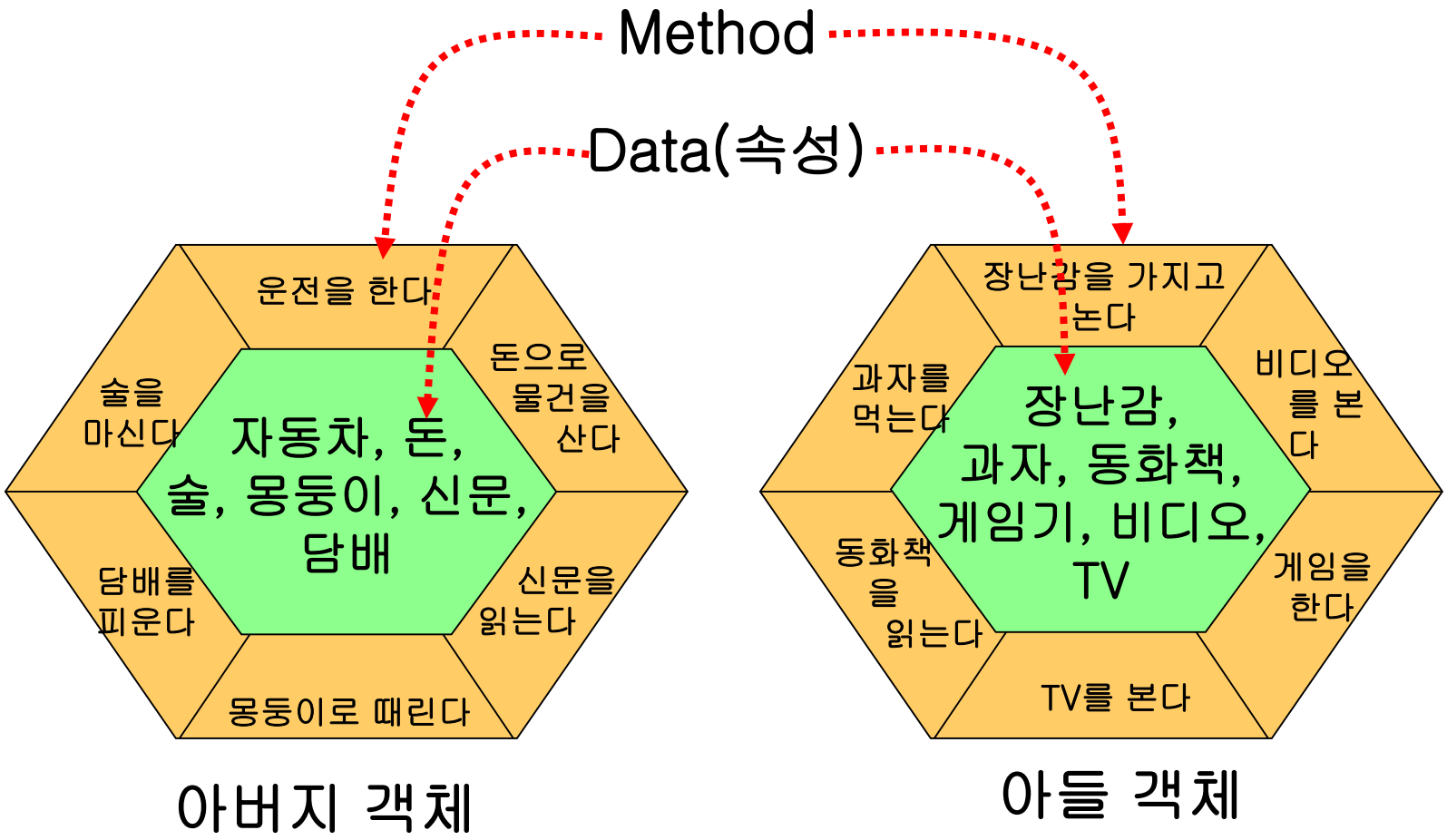
자동차를 이루는
부품들



워드 프로세서를
이루는 객체들

Object Oriented 개념

- Object Diagram
  - 객체는 프로그래머에 의해 모델링



Object Oriented 개념

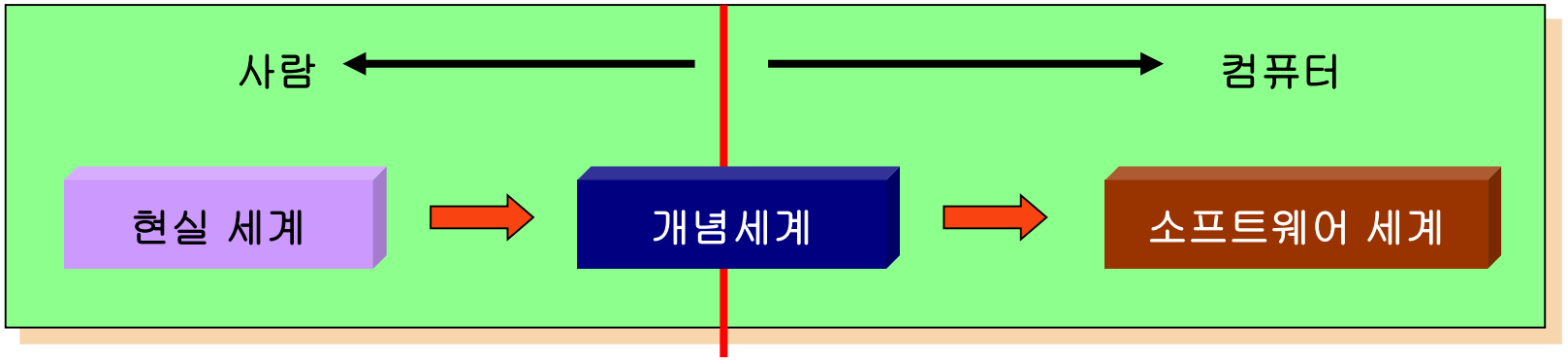
- Object Oriented 방법론의 이점
  - **실 생활의 개념과 거의 동일함**
  - Programmer**가 작성해야 할** Program**의 양이 줄어 듬**
    - **각** Object**들마다 완벽한** Modularity(**모듈화**)**를 제공하기 때문에 코드 재사용**(Code Reuse)**을 극대화 할 수 있음**
  - Maintenace**가 편리**

Object Oriented 개념

- Object Oriented의 중요개념
  - Object(객체)
  - Class(클래스)
    - Object(객체)를 추상화하여 독립적으로 다룰 수 있도록 한 자료형
  - Inheritance(상속)
    - 재사용성(Class 확장의 의미)
  - Encapsulation(캡슐화)
    - 모듈화, Information hiding(정보 은닉)
  - Polymorphism(다형성)
    - method overriding & method overloading
  - Message(메시지)

Object

- Object란 사람이 오감으로 느낄 수 있는 현실 세계에서 우리가 다루는 모든 사물 또는 추상적인 개념
- 문제 영역의 현실 세계에 존재하는 사물의 집합을 추상화하여 효율적으로 정보를 관리하기 위해 의미를 부여하고 분류하는 개념적인 단위



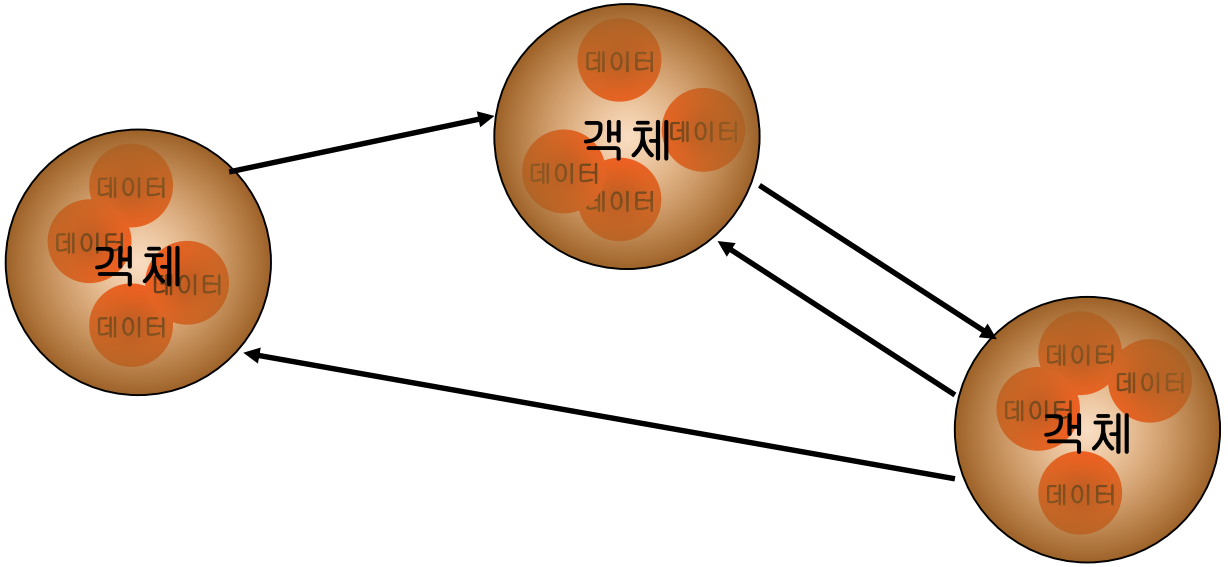
- Object = Data + Method(처리 절차)
  - Data (명사)
    - 상태(state), 속성(attribute)
  - Method(동사)
    - 행동(behavior), 함수(function), Message Interface

Object

- Real World(현실 세계)
  - Object가 존재하는 현실 세계
  - 우리가 일상에서 보는 사물을 일컬음
- Conceptual World(개념 세계)
  - Object가 어떻게 구성됐는지 분석하는 단계
  - 현실 세계에 존재하는 Object의 상태(state)와 행동(behavior) 정보를 추려냄 (abstract)
- Software or Computing World
  - 개념 단계에서 추상화 해 생성된 상태(state)와 행동(behavior)을 Computer 언어로 표현
    - 상태(state)
      - Programming하기 위한 변수
    - 행동(behavior)
      - Method(or Function)로 표현

Object

- 상태(Status)와 행동(Behavior)
  - State(명사)
    - Memory 내부에 정의된 Data – “Data”
  - Behavior (동사)
    - CPU를 통한 명령 실행의 집합 – “Code 집합 (Method)”



Object

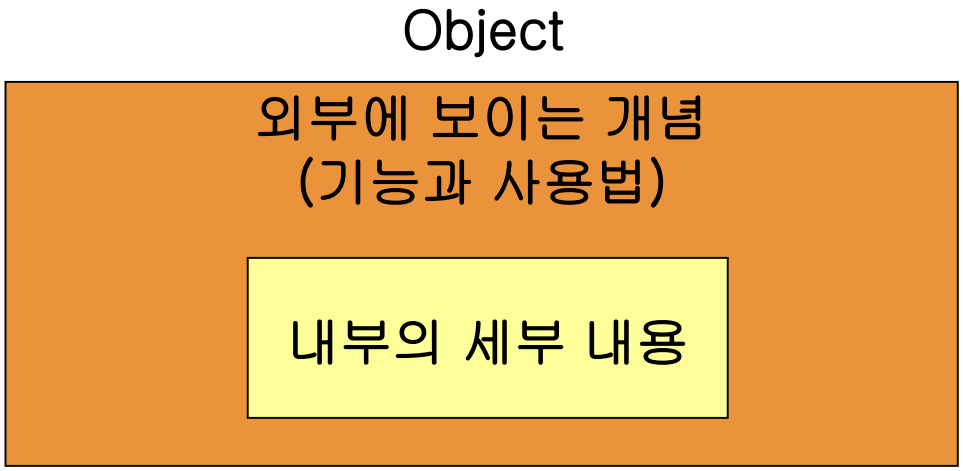
- 표현의 한계
  - 우리가 사물을 정의할 때 그 사물이 갖는 모든 요소들을 다 나열하여 정의 할 수는 없음
    - 예) 여러분들 스스로 “의자”를 정의해 보아라
    - 같은 사물에 대하여 여러 가지 정의가 존재할 수 있음
    - 예) 의자
      - 의자란 둔부가 바로 닿는 Seat와 등받이가 갖추어진 것
      - 가죽 의자는 Seat를 가죽으로 만듦
- 사물을 정의할 때 한가지 중요한 요소가 작용
  - 그 사물이 갖는 특징을 찾아 다음 두 가지 측면에서 표현
    - 사물의 성질(의자의 재질, 다리의 개수, 등받이 여부...)
    - 사물의 기능(의자의 기능)
    - 이를 추상화(abstraction) 과정이라고 함

Object

- 추상(Abstract)
  - “인간”의 개념을 떠올려 볼 수 있음
    - 추상적인 개념
    - 공통적인 특징 인식
  - “인간”의 경계?
    - 언제부터 “인간”으로 인식하는가?
    - 엄마 배속에 있어도 “인간”? 10개월, 9개월… 상실기, 수정 직후
- Abstraction(추상화)
  - Software의 목적에 맞게 개념 정의
    - 필요한 부분만 추상화
    - 본 Program에서 이 클래스는 ~라고 정의함
  - 클래스 정의

Object

- Object와 Abstraction
  - 외부에 나타나는 개념은 추상화(abstraction)를 통해 형성됨
  - Abstraction
    - 중요하지 않은 세부 내용을 제거함으로써 핵심적인 내용만을 추출함



Object

- Object에 의해 현실 세계를 Abstraction하는 효과
  - 안정된 Model을 구축
    - 기능 중심에서 물체 중심
    - 분석의 많은 측면 중에서 물체의 구조가 가장 안정된 측면
    - 물체 중심의 접근 체계는 기능의 확장과 변경에 쉽게 대응
  - 실 세계를 자연스럽게 표현
    - 물체의 구조에 관심
    - 영역 분석에 의해 객체를 재사용
    - 실 세계와 비유를 이용하여 필요한 기능과 행동을 유추
  - 분석의 초점이 명확
    - 현실 세계의 중요한 측면만 Modeling

Object

- Data 추상화
  - 코끼리가 무엇인지를 모르는 인디언들이 있다. 그러나 그 가운데에는 세상 물정에서 아주 밝은, 그래서 코끼리를 본 적이 있는 인디언이 한 명 있다. 그 인디언이 나머지 인디언들에게 코끼리가 무엇인지를 설명한다.
  - "코끼리는 말이야. 일단 발이 4개야. 그리고 코가 하나 있는데, 그 코의 길이가 한 5미터 정도는 되지. 그리고 그 코를 이용해서 목욕을 하기도 하고, 또 물건을 집기도 해. 크기는 말야. 글쎄 무게로 따지면 1톤은 족히 넘을껄."

Object

- Data 추상화
  - 이 말을 들은 인디언들은 코끼리가 무엇인지를 정의하기 시작한다
  - 1) 발이 4개
  - 2) 코의 길이가 5미터
  - 3) 몸무게는 1톤 이상
  - 4) 코를 이용해서 목욕을 함
  - 5) 코를 이용해서 물건을 집기도 함
  - 코끼리가 무엇인지에 대한 정의가 내려졌다. 특징 1, 2, 3은 코끼리의 데이터적인 측면이고, 특징 4, 5는 코끼리의 기능적인 측면
  - 비록 부족하기는 하지만 코끼리가 무엇인지에 대한 정의가 완전히 내려진 것이다. 이것이 바로 데이터 추상화

Object

- Data 추상화
  - 즉 현실 세계의 사물을 데이터적인 측면과 기능적인 측면을 통해서 정의하는 것이 바로 데이터 추상화
  - 코끼리에 대한 정의를 내리긴 했지만 지극히 관념적이고 추상적이지 않은가. 그래서 데이터 추상화라고 함
  - 데이터 추상화에서의 데이터는 사물의 데이터적인 측면과 기능적인 측면을 동시에 일컫는 것

Object

- Object의 종류
  - 세상 모든 것이 객체(Object)
  - Object란 사람이 오감(五感)으로 느낄 수 있는 현실 세계에서 우리가 다루는 모든 사물
  - Object란 효율적으로 정보를 관리하기 위하여, 사람들이 의미를 부여하고 분류하는 논리적인 단위
  - Object란 우리의 실 세계에 존재하는 자동차, 사람, 건물 등 이러한 모든 것을 Modeling 할 수 있으며 Software적 표현으로 정의 할 수 있음

Object

- Object의 종류

현실 세계에 존재하는 물체	추상적인 개념의 집합
▪ 사람(학생, 교수, 교직원, 친구) ▪ 각종 건물이나 기관 ▪ 작성된 그림이나 도표 ▪ 각종 업무 관련 서류나 양식 ▪ 소프트웨어 패키지 화면 ▪ 화면상의 아이콘이나 필드	▪ 학생의 성적 ▪ 제조 업체의 처리 공정 ▪ 기계 장치의 작동 메커니즘 ▪ 비행기 좌석 예약

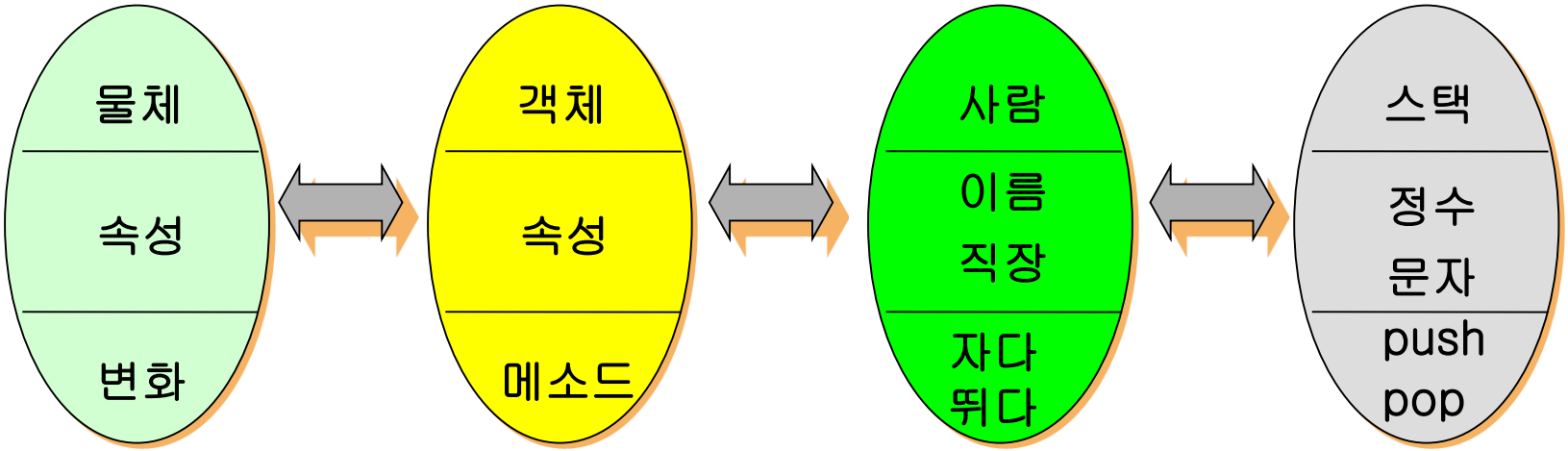
- Object 예
  - 사람이라는 객체는 “이름”이나 “직장” 또는 “주민등록번호” 같은 속성을 갖고 있으며, “잠을 자거나” “밥을 먹거나” “뛰는” 등의 메소드로 구성
  - Data를 특정 구조에 넣거나 빼내는 구조인 Stack은 원하는 Data를 속성으로 갖고 있으며, 일정한 Data를 삽입하는 push() 메소드와 추출하는 pop() 메소드를 가짐

Object

- 좋은 Object가 되기 위한 3가지 조건
  - State(상태)를 유지
    - Object의 상태는 각 Object가 가지고 있는 변수들의 현재 값에 의해 결정
    - 예) ‘A’라는 정수형 변수가 ‘3’이란 값을 가지고 있으면, 그 정수는 ‘3’이란 상태를 가진다고 표현
  - Behavior(기능)을 포함
    - 기능은 Object가 어떤 일을 수행할 수 있는 능력을 의미
    - 기능은 Object가 수행할 수 있는 메소드에 의해 결정
  - Identifier(식별자)를 포함
    - Object는 모양과 기능이 비슷해도 다른 Object와 유일하게 구별할 수 있어야 함

Object

- Object의 구성요소



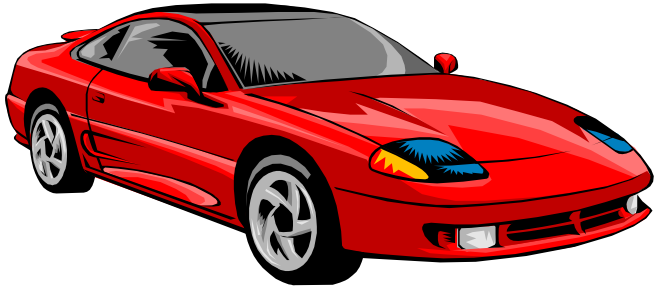
- Object Oriented Language에서의 모든 Object는 반드시 Class로부터 생성됨
- Object들간에 Message를 통하여 정보를 교환 함으로서 일을 수행

Object

- Object의 구성요소
  - 현실 세계의 Object를 Program에 반영하려면, 실제 Object를 Program에 적용할 수 있는 형태로 변형시켜야 함(추상화 : abstract)
  - 추상화는 어떤 Object에서 명사와 동사를 추출해내는 것을 의미
  - Object를 명사와 동사로 추출하는 예
    - Object 명 : Car(자동차)

Object

- 자동차 Object



- 자동차의 색상
- 자동차의 속도
- 방향
- 차량 번호
- 차종

상태, 속성(가공할 정보)

메소드, 서비스, 행위
(제공하는 기능)

- 달린다
- 방향을 바꾼다
- 멈춘다
- 가속한다

Object

- Object의 구성요소
  - Object를 명사와 동사로 추출하는 예
    - Object 명 : Person(사람)
      - 명사
        - 이름, 나이, 주소, 핸드폰 번호, 몸무게, 키, 신발 size, ……
      - 동사
        - 밥을 먹다, 잠을 잔다, 걷는다, 운동을 한다, …
    - Object 명 : Account(계좌)
      - 명사
        - 계좌 번호, 비밀번호, 잔고, 이율, 이체 한도, …
      - 동사
        - 입금하다, 출금하다, (잔고를) 조회하다, …

Object

- Object의 구성요소
  - Object는 자신의 고유한 속성(state)과 행위(behavior)변화로 구성
  - State (명사)
    - 속성을 나타내는 Data
  - Behavior (동사)
    - Data를 조작하고 처리하는 절차를 기술하는 메소드(method)

- Object의 이점
  - 모듈화(modularity)
  - 정보 은닉(information hiding)
  - 재사용성(reusability)

Class

- Object Oriented Programming에서는 모든 “자료”와 “동작”이 Object로 표현
  - Program 작성은 주어진 문제를 분할하여 잘 정의된 Object로 기술하는 것
  - Object를 Program 상에서 기술하기 위해서는 각 Object의 속성을 정의하는 수단이 필요

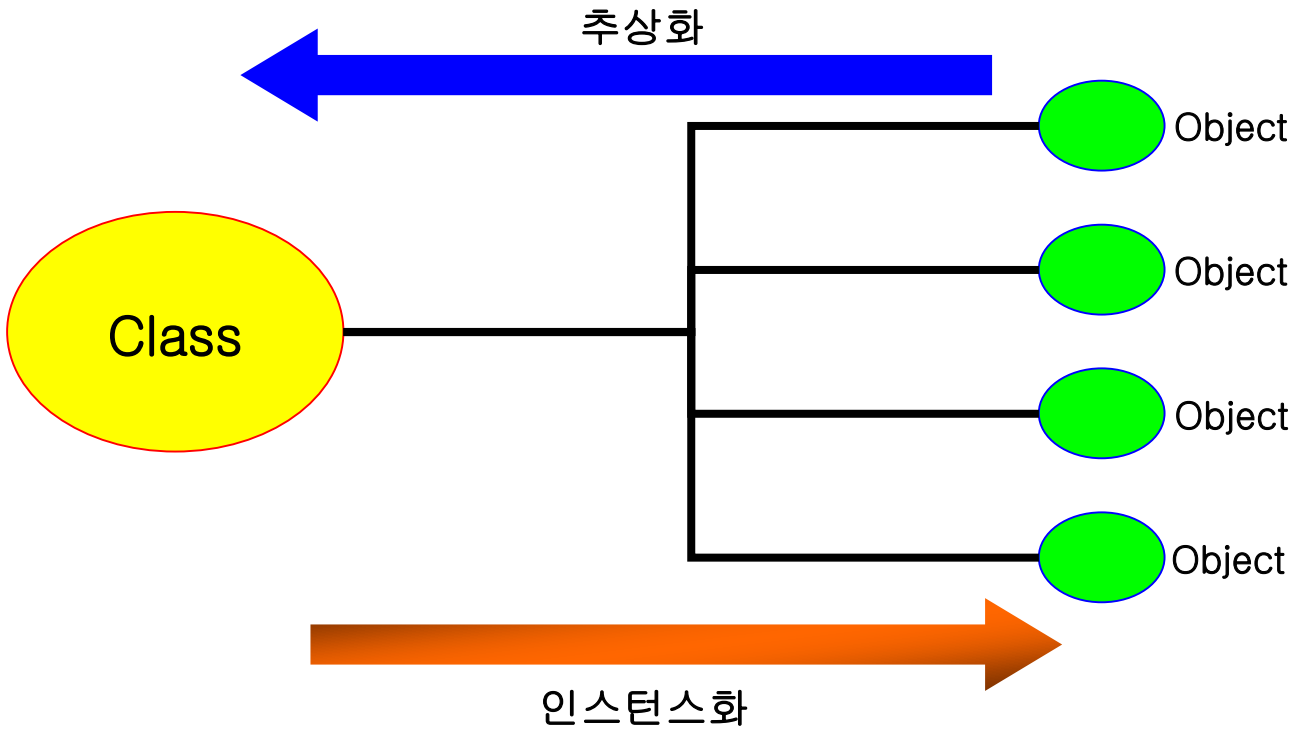
- Object의 속성을 정의하는 수단 : 클래스(Class)
  - 각 Object가 어떠한 구조를 갖고 있으며, 그 Object가 받아들이는 Message는 어떠한 것이 있는가, 또한 그 Object가 Message를 받았을 때 어떤 방식으로 처리될 것인가 등의 내용을 기술
  - 모든 Object는 사용되기 전에 먼저 Object의 속성을 기술하는 Class를 정의한 후 이 Class의 형으로 선언

Class

- Object Oriented Program을 작성하다 보면 같은 기능을 하는 Object를 여러 개 사용할 필요성이 요구됨
  - 예) 50명의 학생과 관련된 응용 Program(성적 처리 프로그램)을 작성하기 위해서는 50명의 학생 객체가 필요로 하게 됨
  - 50명의 학생들의 같은 속성(이름, 성별, 학과 ,학년)과 메소드(수강신청,시험보기, 성적조회)를 가지고 있음
- Object Oriented Language에서는 동일한 속성과 메소드를 가진 Object를 생성하기 위해 Class라는 형판을 제공
  - Class는 하나의 Class로부터 여러 개의 Object를 생성하기 위해 사용되는 형판
- JAVA로 Programming 한다는 것은 곧 Class를 만들어서 사용한다는 의미

Class

- 공통된 속성과 메소드들을 갖는 모든 Object들을 모아 놓은 것
- 어떠한 Object도 그 Object를 포함하는 Class가 존재
  - Object는 해당 Class의 Instance



Class

- Class와 Object는 '붕어빵 틀'과 '붕어빵'의 관계
- 혹은 '제품의 설계도'와 '제품'의 관계라고 말할 수도 있음

클래스에 비유할 수 있는 것들	 붕어빵 틀	 제품 설계도
객체에 비유할 수 있는 것들	 붕어빵	 제품

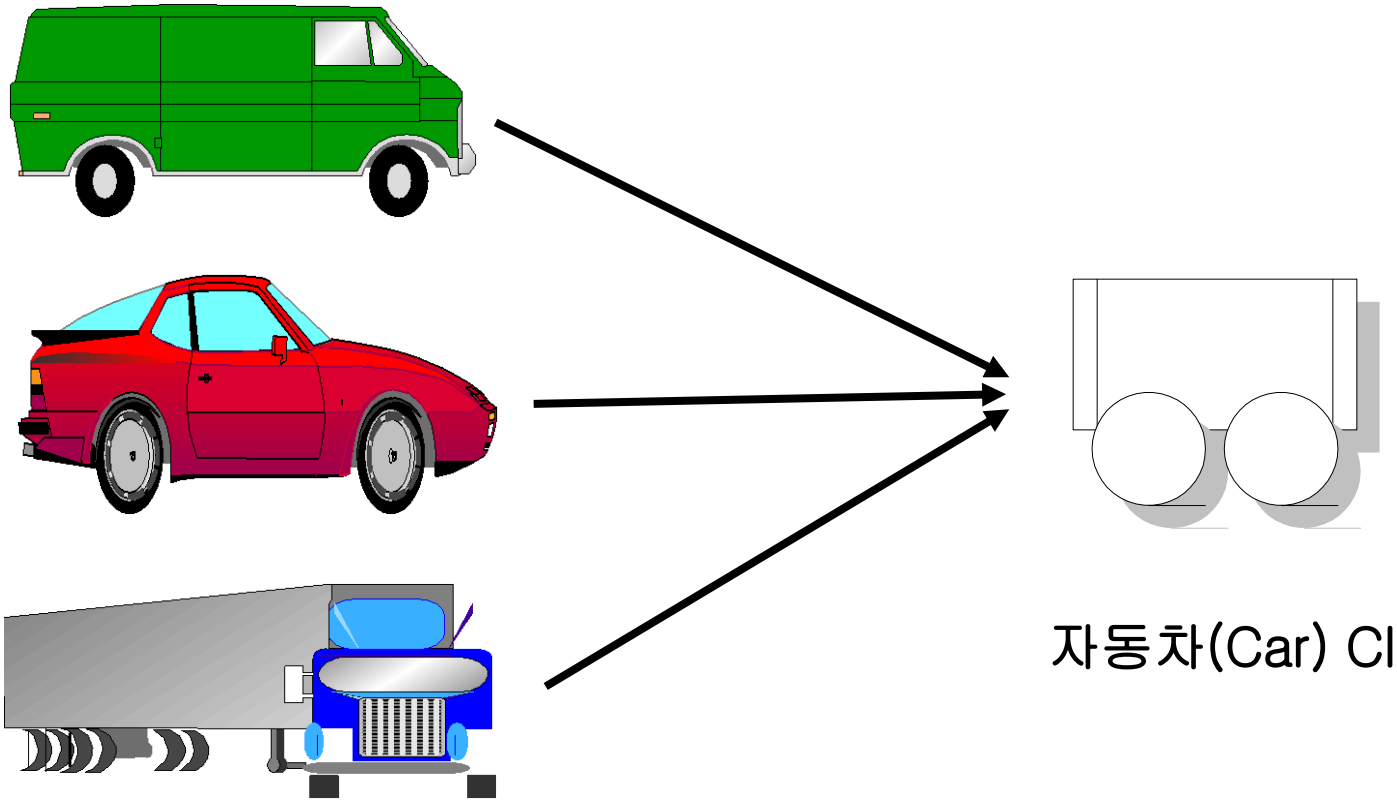
Class

- Object와 Class 비교

Class	Object
객체들을 생성해주는 본형	어떤 Class의 Instance
객체들의 특성들을 기술해 놓은 기술서	객체마다 각각의 상태를 갖고 있음
개념적, 가상적	물리적, 실제적
Program Text상에서만 볼 수 있음	실행 시에만 존재
설계 단계에서 관심사	실행 단계의 관심사

Class

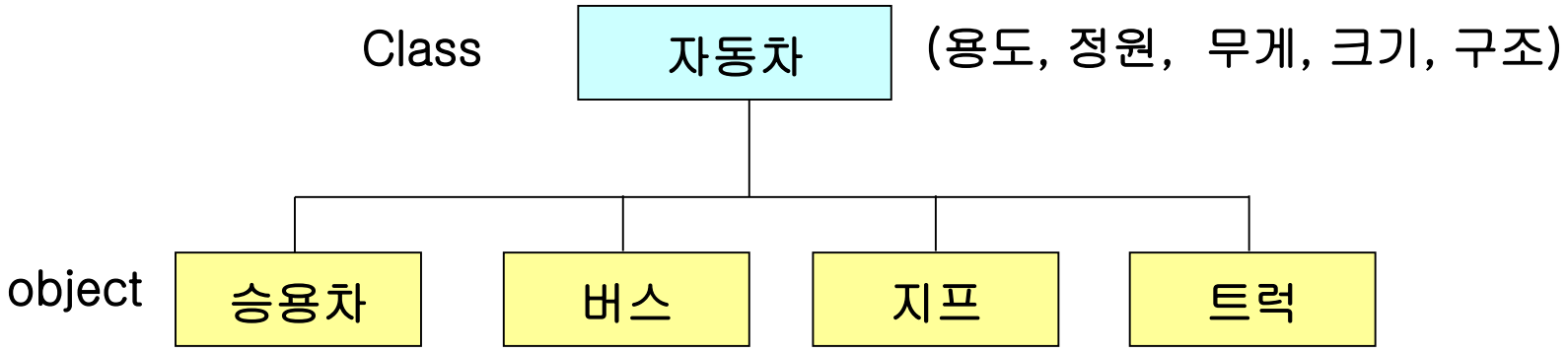
- Class와 Instance의 예
  - Class는 우리 인간의 사고과정을 통해 한 단계 추상화 시킨 일반 명사이고, Object는 세상에 유일하게 존재하는 모든 고유 명사



자동차(Car) Class

Class

- 예) 자동차라는 Class



  - 자동차의 속성은 용도, 정원, 무게, 크기, 구조 등으로 정의되며 이 성질에 따라 여러 종류의 자동차가 존재
  - 이때 속성의 정의는 자동차라는 Class가 되고, 각 Class의 속성의 값을 초기화 하여 여러 종류의 자동차 Object들이 선언

Class

- 동일한 속성과 메소드를 가진 Object를 생성하기 위한 Prototype(형판)
  - Class : 속성(Data의 구조) + Method
  - Instance : Class로부터 생성된 객체
  - Object : Data + Method
- Class에 포함된 변수
  - 멤버(member), 애트리뷰트(attribute)
- Class 내에 포함된 Routine
  - 행동(behavior), 메소드(method), 멤버 함수(member function)
- Class의 이점
  - 재 사용성(reusability)

Class

- Class 생성
  - Object는 항상 Class로부터 생성
  - Class는 Object를 생성하는 형판(template)
  - Class는 두 개의 구성 요소인 자료 구조(Field)와 연산(Method)을 가짐
  - Class로부터 생성된 Object를 Instance라고 함
    객체 = instance
  - 정보 처리의 주체는 Class가 아니라 Object
  - Object Oriented Programming의 시작은 Class의 생성
  - Class로부터 Object의 생성 예

```
학생1 = new 학생(홍길동, 남자, 컴퓨터공학과, 2학년);
```

객체 이름

객체 생성 명령어

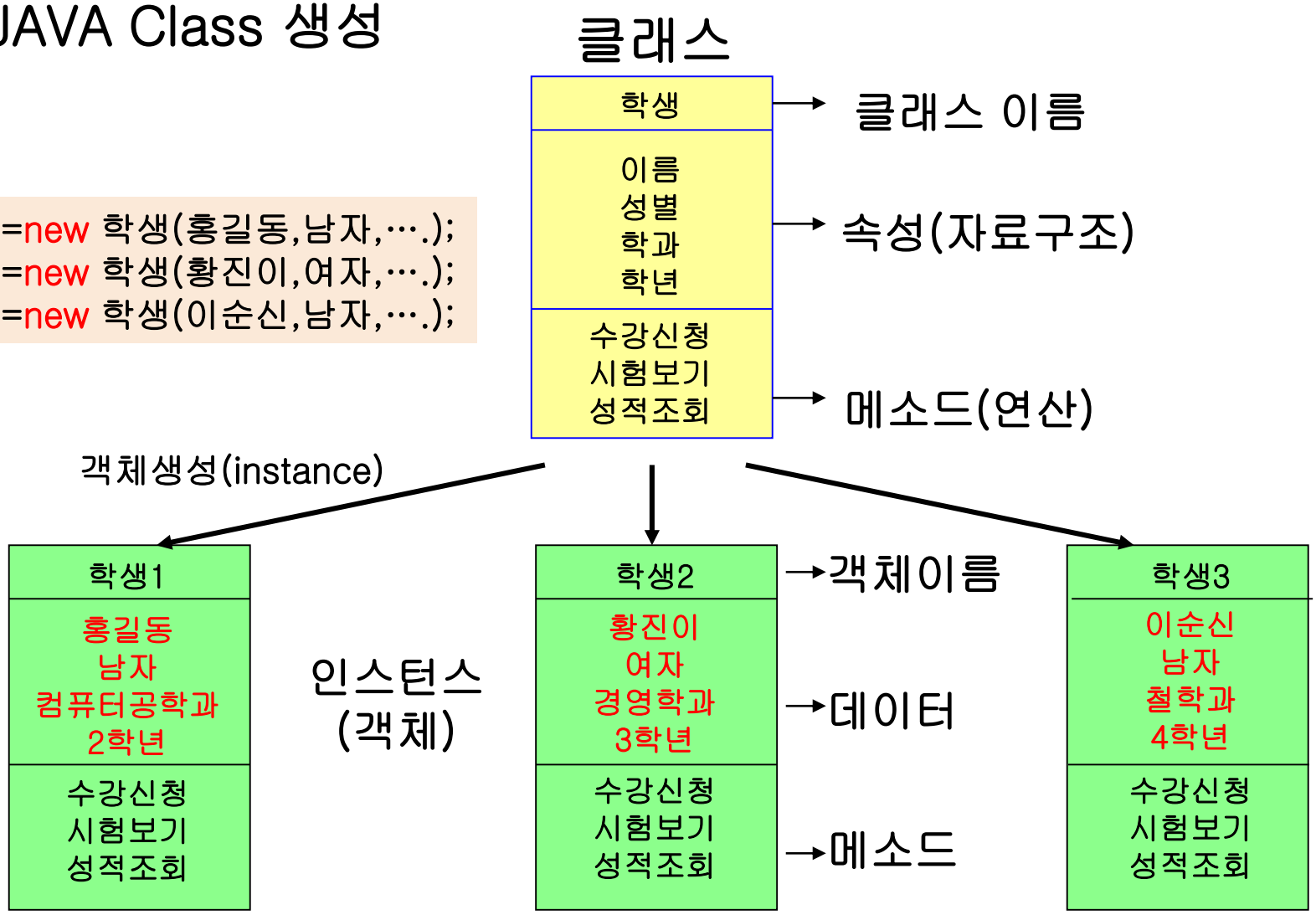
클래스 이름

매개 변수 데이터

Class

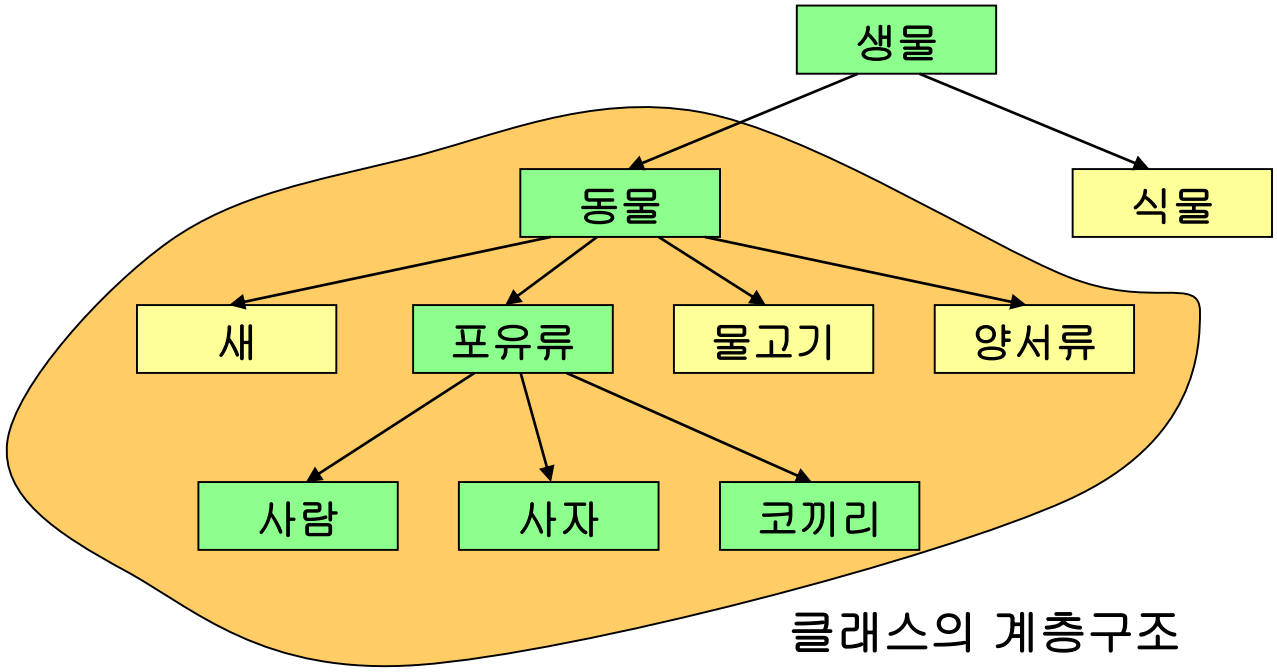
- JAVA Class 생성

학생1=new 학생(홍길동,남자,….);
학생2=new 학생(황진이,여자,….);
학생3=new 학생(이순신,남자,….);



Class

- Class Hierarchy
  - Class는 계층 구조를 이룸
  - 새로운 Class를 생성할 때 기존 Class의 하위 Class로 선언할 수 있음
  - 새로운 Class에 속성이나 메소드를 추가하여 기존 Class를 확장할 수 있음



클래스의 계층구조

일반화
- 간단한 공통 속성

특수화
- 상위 속성 상속
- 개별 속성 추가
- 많아진 속성