Object Oriented 개념

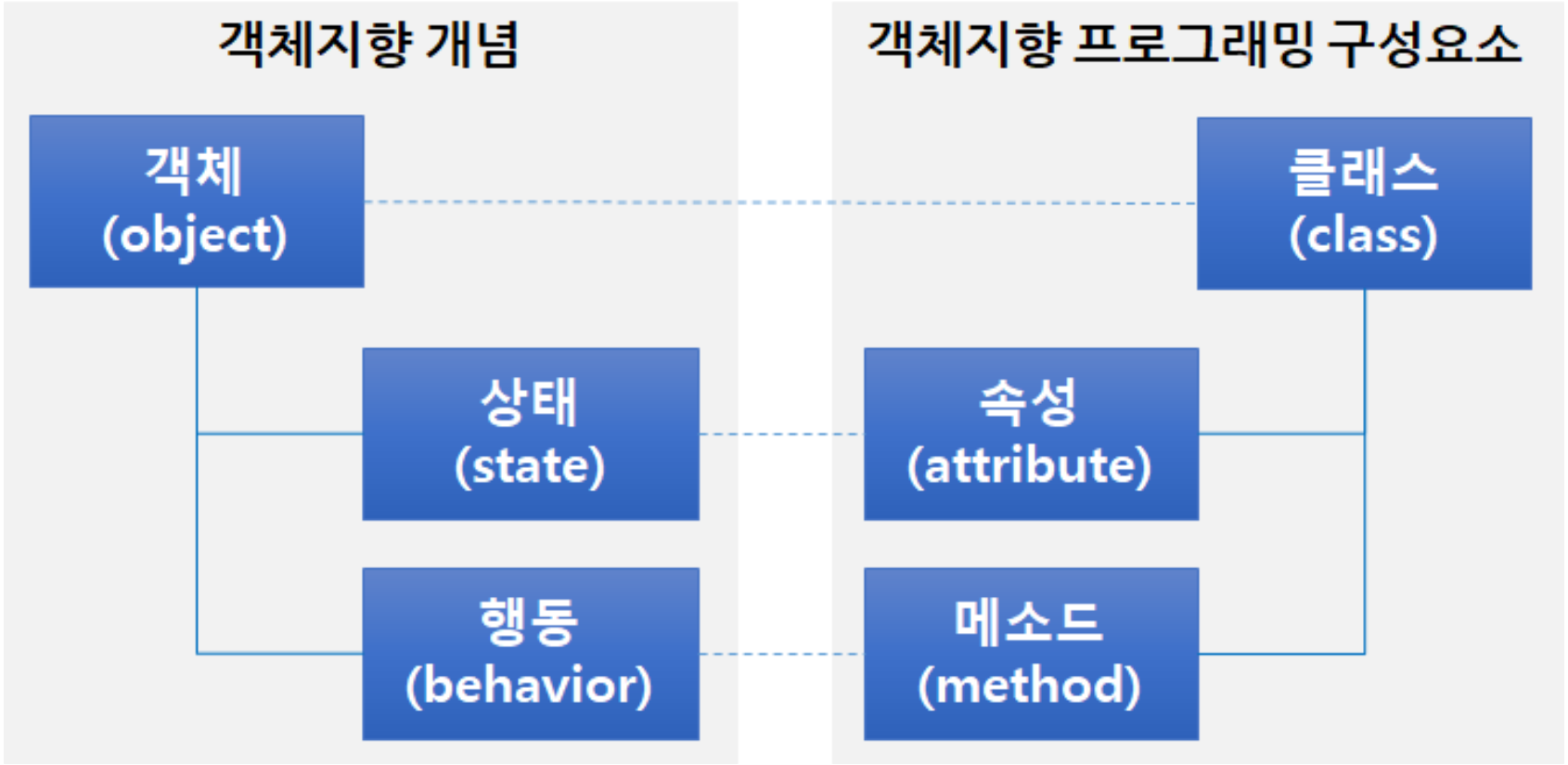
경복대학교
소프트웨어융합과
배 희호 교수

Object Oriented Language

- Alan Curtis Kay(앨런 케이)
  - 미국의 컴퓨터 과학자
  - 이반 서덜랜드와 함께 스케치 패드 개발(1960년대 유타 대학교)
  - 제록스 파크(PARC)에서 Smalltalk 개발 (70년대)
  - Object Oriented Programming을 개척한 공로로 튜링상 (2003)

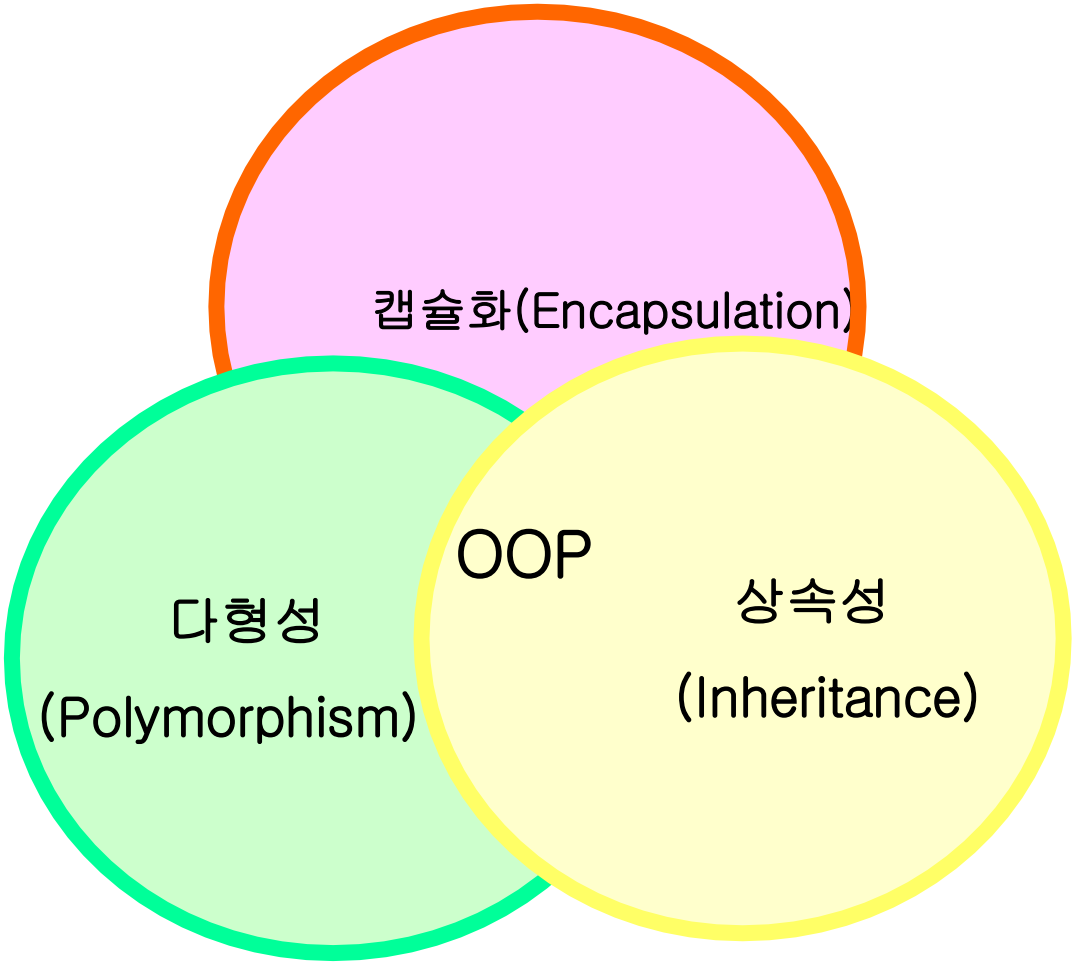
Object Oriented Language

- 구성 요소



Object Oriented Language

- Object Oriented 개념 특성



Object Oriented Language

- 목적
  - Software의 생산성 향상
  - Computer 산업 발전에 따라 Software의 생명 주기(Life Cycle) 단축
  - Software를 빠른 속도로 생산할 필요성 증대
    - Object, Encapsulation, Inheritance, Polymorphism 등 Software 재사용을 위한 여러 장치 내장
    - Software 재사용과 부분 수정 빠름
    - Software를 다시 만드는 부담 대폭 줄임

Object Oriented Language

- 객체지향의 4대 특성

특성	내용	예
캡슐화	외부에서는 생성한 Object가 어떤 Method와 Field로 일을 수행하는지 몰라도 됨	예) getter/setter
상속	자식 Class는 부모 Class를 물려받으며 확장 가능 (재사용)	예) extends 키워드
추상화	공통된 속성/기능을 묶어 이름을 붙임 (단순화 및 Object Modeling)	예) Class, Object
다형성	부모 Class와 자식 Class가 동일한 요청을 다르게 처리할 수 있음 (사용 편의성)	예) Override, Overload

Object Oriented Language

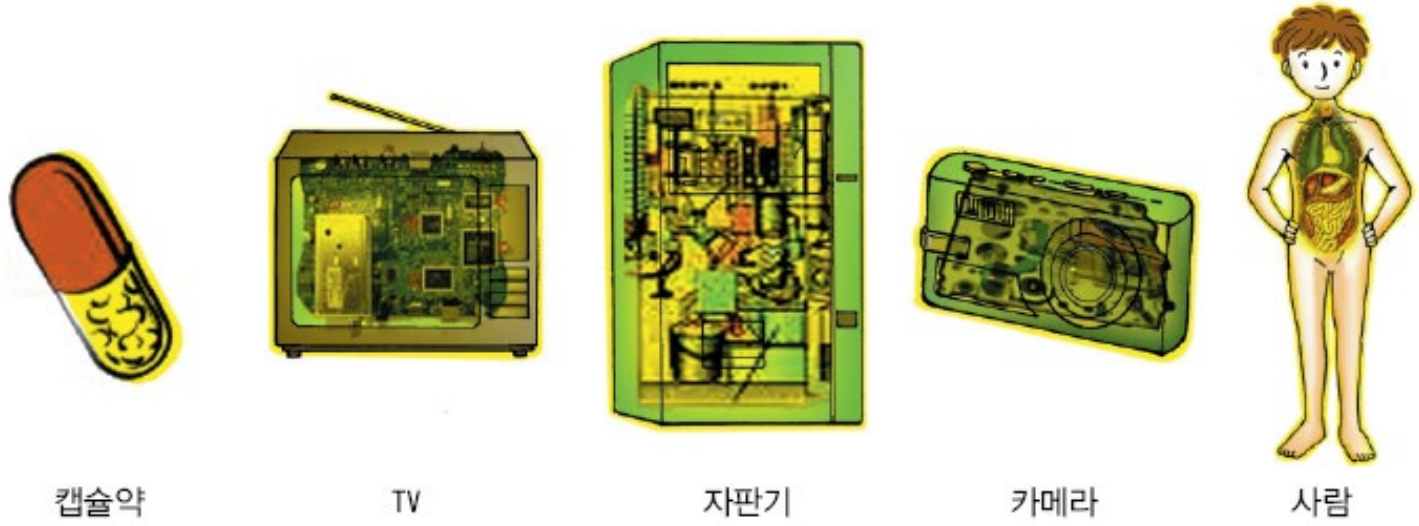
- Encapsulation(캡슐화)

Object Oriented Language

- Encapsulation(캡슐화)
  - Class 내부에 여러 Attribute(Data)와 여러 Method(Data에 대한 조작)를 함께 묶는 것을 의미함
  - Encapsulation는 Class 내부의 Attribute나 Method를 외부에 노출하지 않고 보호하는 것을 의미
  - 이렇게 Encapsulation은 묶는 것과 보호하는 것을 생각할 수 있음
  - 여러 Attribute와 여러 Method를 함께 묶어 Class로 취급하는 것과 Class 내부를 외부에서 접근하지 못하도록 보호하는 것이 바로 Encapsulation

Object Oriented Language

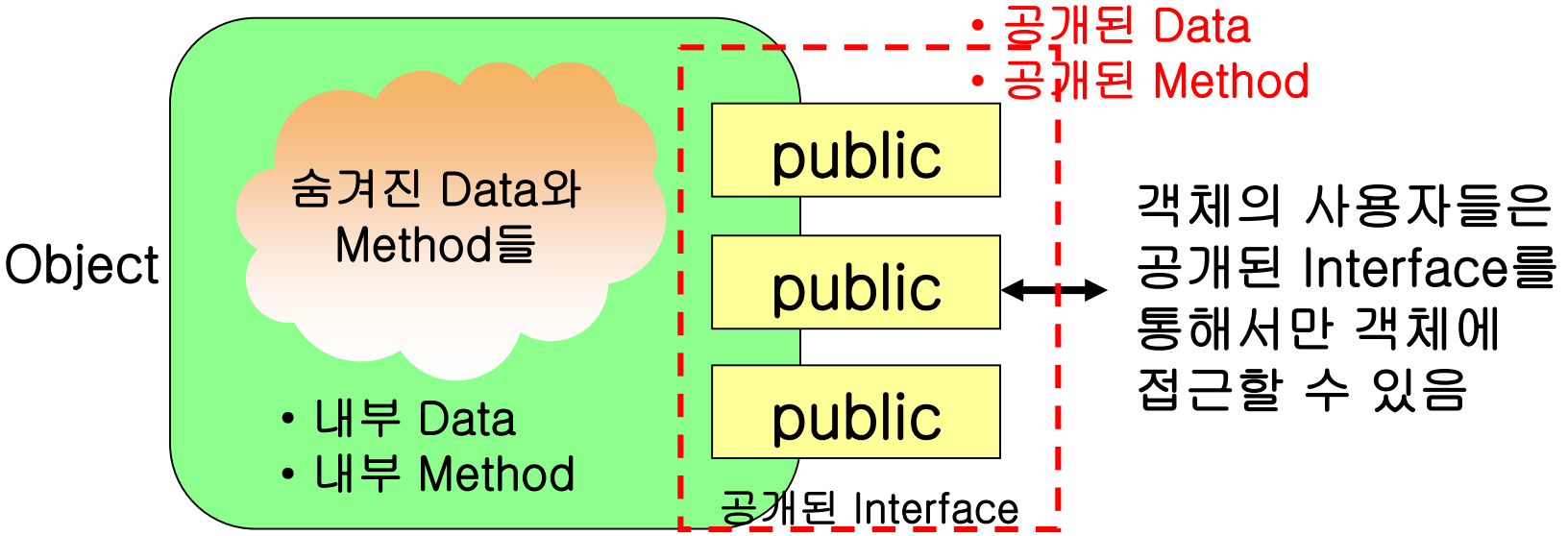
- Encapsulation(캡슐화)
  - Method(함수)와 Data를 Class내에 선언하고 구현
  - 외부에서는 공개된 Method의 Interface만으로 접근 가능
    - 외부에서는 비공개 Data에 직접 접근하거나 Method의 세부적 구현 방법을 알 수 없음
  - Object 내 Data에 대한 보안, 보호, 외부 접근 제한



실 세계의 캡슐화

Object Oriented Language

- Encapsulation(캡슐화)
  - Object를 작성할 때 숨겨야 하는 정보(private)와 공개해야 하는 정보(public)를 구분하여 접근 권한 제어 가능
  - Object가 보유하고 있는 Data와 연산에 대해 외부에 가시성을 적절히 제공
  - Object의 사용자는 기능만 알고 사용하며 어떻게 처리되는지는 은폐됨(Information Hiding)



Object Oriented Language

- Encapsulation을 통한 장점
  - Module화
  - Information hiding(정보 은닉)
    - Object에 포함된 정보의 손상과 오용을 막을 수 있음
    - 연산 수행 방법이 바뀌어도 연산 자체에는 영향을 주지 않음
    - Data가 변해도 다른 Object에 영향을 주지 않아서 독립성이 탁월함
    - 처리된 결과만 사용하므로 Object의 확장성과 이식성이 뛰어남
    - 연산만 알면 Object를 이용하여 새로운 System을 설계할 수 있음

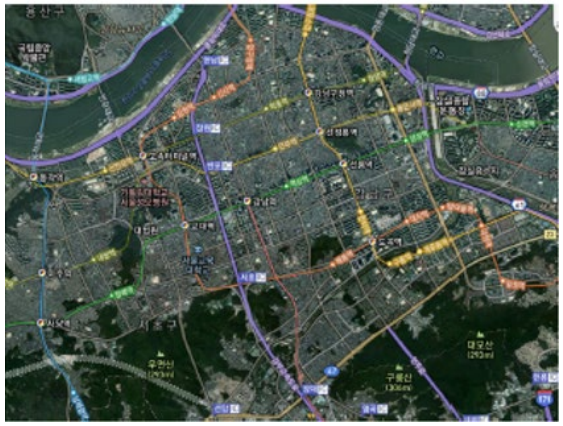
Object Oriented Language

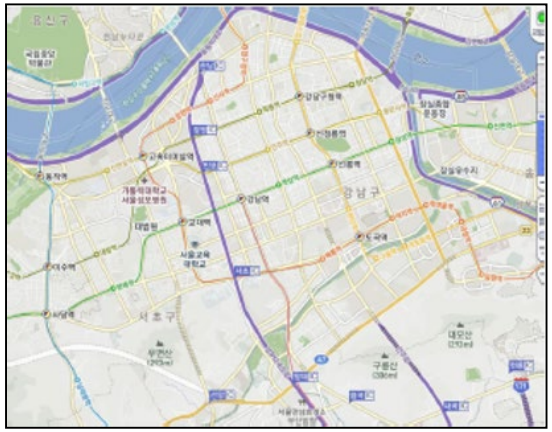
- Encapsulation의 효과
  - Object의 이용자와 제공자의 역할 분담을 명확화
    - Object 이용자
      - Object에 정의된 절차의 사용 방법만 알면 됨
      - Object의 자료 구조를 이해 할 필요가 없음
      - 이용자에게 이용이 쉬움
    - Object 제공자
      - 내부 자료 구조를 변경해도 이용자에게 영향을 미치지 않음
      - 제공자에게 자료 구조 변경이 용이

Object Oriented Language

- Abstraction(추상화)
  - 실 Object를 Modeling할 때, 문제 해결에 필요한 부분만을 Modeling하는 기술
  - 복잡함 속에서 필요한 부분만 표현하는 것
  - 예) 캐리 커쳐

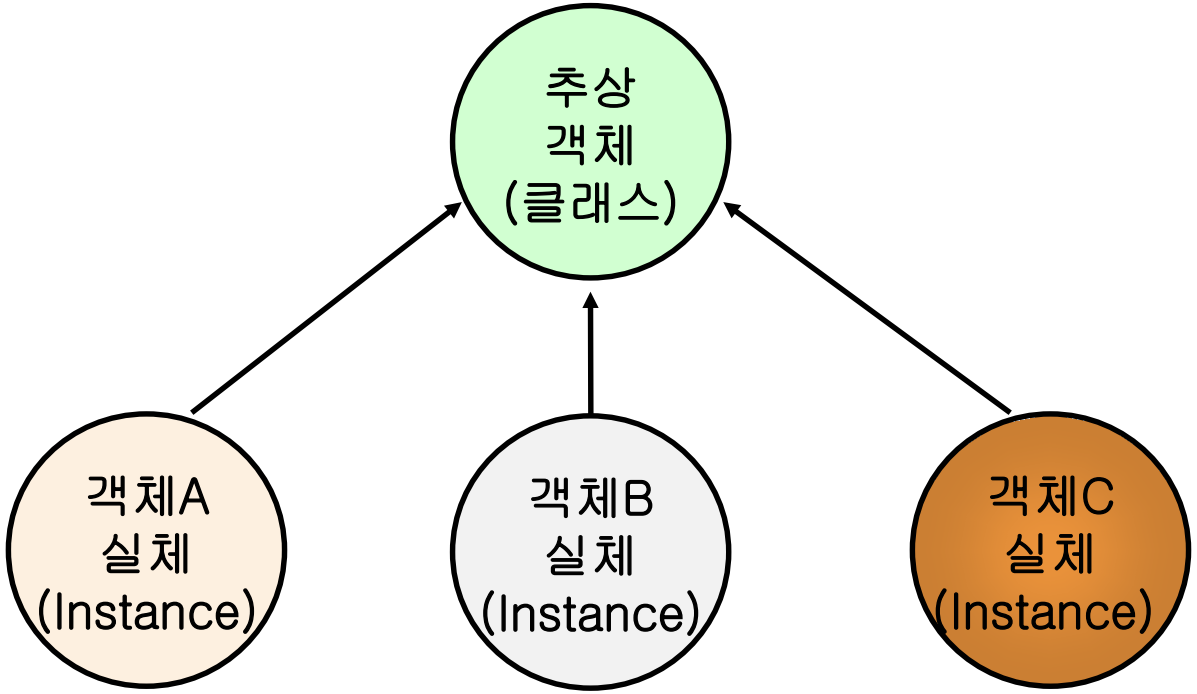
- 단순히 지하철을 이용하면 어떤 지도가 가장 적합할까요?





Object Oriented Language

- Abstraction(추상화(抽象化))
  - 抽(뽑아낼 추), 象(코끼리 상, 모양 상)
  - 대상으로부터 모양을 뽑아내는 것



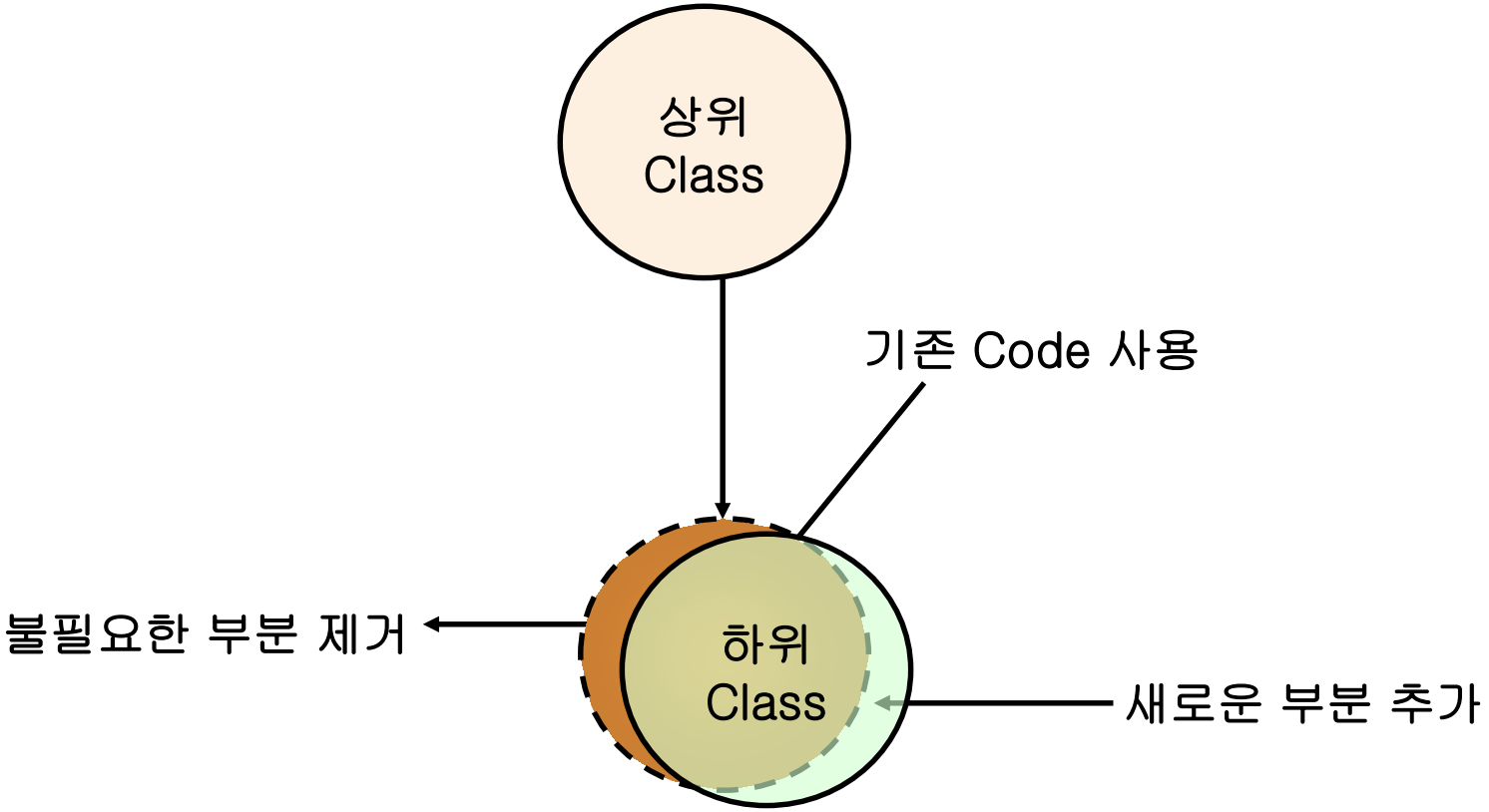
추상 클래스는 표현 대상들의 공통적인 특징을 서술(description)한 것

Object Oriented Language

- Abstraction(추상화)
  - Object와 Procedure들의 공통의 특징들을 골라내는 과정으로 Object의 자세한 성질을 숨기고 그들의 일반적인 성질을 나타낸다는 것
  - 일반적으로 Class는 Class로 표현할 Sub Class(또는 Object)의 공통적인 성질과 행위를 일반화하여 Design되게 되며 그로부터 생성된 Object는 자신의 고유의 성질을 가지게 됨
  - 예) Programmer가 거의 같은 작업을 수행하는 2개의 함수를 하나의 함수로 합병할 수 있음을 나타내기 위해 추상화를 사용할 수 있음. 추상화는 Software 공학에서 가장 중요한 기법 중 하나로서, 2개의 다른 기법인 캡슐화 및 정보 은폐와 매우 밀접한 관련이 있음

Object Oriented Language

- Inheritance(상속)

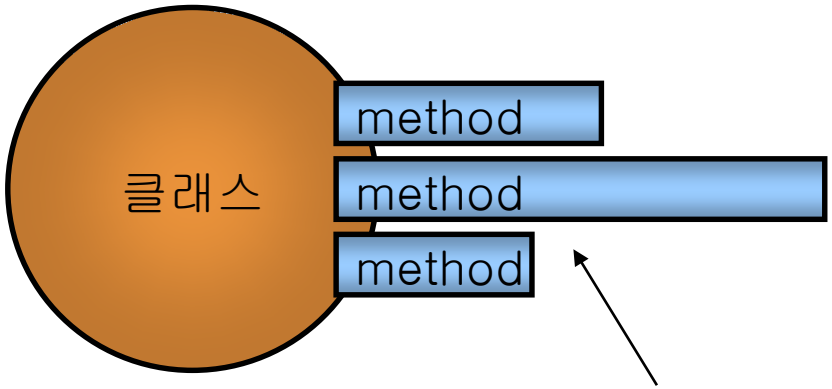


Object Oriented Language

- Inheritance(상속)
  - 하나의 Class Object가 정의되었을 때, 차후 정의되는 어떠한 Sub Class라도 일반 Class들의 정의 중에서 하나 또는 그 이상의 정의를 물려받을 수 있다는 개념
  - Programmer에게 있어서 상속이란 개념은, 만약 그것이 Sub Class가 속한 Class의 포괄적인 Attribute라면, Sub Class 내의 하나의 Object는 자기 자신만의 Data 또는 Method에 관한 정의를 가지고 다닐 필요가 없다는 것을 의미
  - 이것은 Program 개발 속도를 높여줄 뿐 아니라, 정의된 Sub Class Object들에 대한 유효성이 본래부터 확실하다는 것을 보장

Object Oriented Language

- Polymorphism(다형성)



이름은 같으나 인자가 다른 메소드

Object Oriented Language

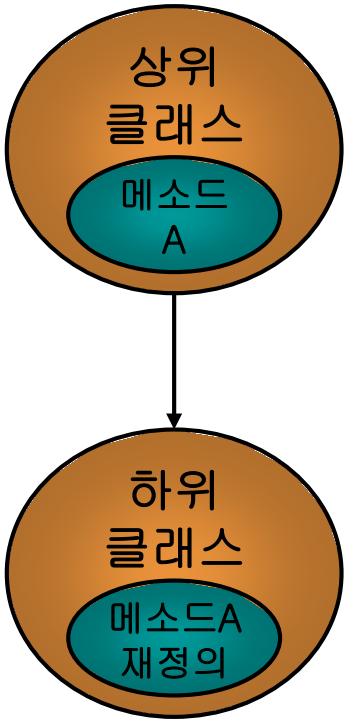
- Polymorphism(다형성)
  - 동일한 이름을 갖는 Method나 연산자가 다양한 문맥(context)에서 겉 모습은 그대로 유지하되 속 모습은 달리하면서 사용될 수 있도록 하는 강력한 기법
  - 예) JAVA 언어에서 1+1이라는 명령문 안에 있는 '+'는 두 개의 정수를 더하는 덧셈 연산자이지만 "abc"+"def"라는 명령문 안에 있는 '+'는 "abc"와 "def"라는 두 개의 문자열을 서로 연결(concatenation)하라는 문자열 연산자임. '+'라는 기호의 겉모습은 유지되고 있지만, 그 연산자가 실제로 수행하는 연산의 내용은 완전히 다름

Object Oriented Language

- Polymorphism(다형성)
  - 다형성이 지원되지 않는 조건에서 '+' 기호를 앞에서와 마찬가지로 여러 가지 목적에 사용한다고 가정하자. 이러한 경우에는 함수에 전달된 인수(parameter)의 데이터형을 확인해서 그것이 '정수'면 '더하기'를 수행하는 함수를 호출하고, '문자열'이면 '문자열 연결'을 수행하는 함수를 호출하는 조건문을 Programmer 스스로 작성해야 함
  - JAVA 언어를 사용해서 Programming을 할 때 Method의 이름은 동일하게 유지하면서 입력되는 인수의 수나 Data형만 살짝 바꿔주는(오버로드(overload)라고 불림) 경우가 있는데, 이러한 방법이 가능한 이유는 바로 JAVA 언어가 다형성을 지원해 주는 '객체지향' 언어이기 때문임

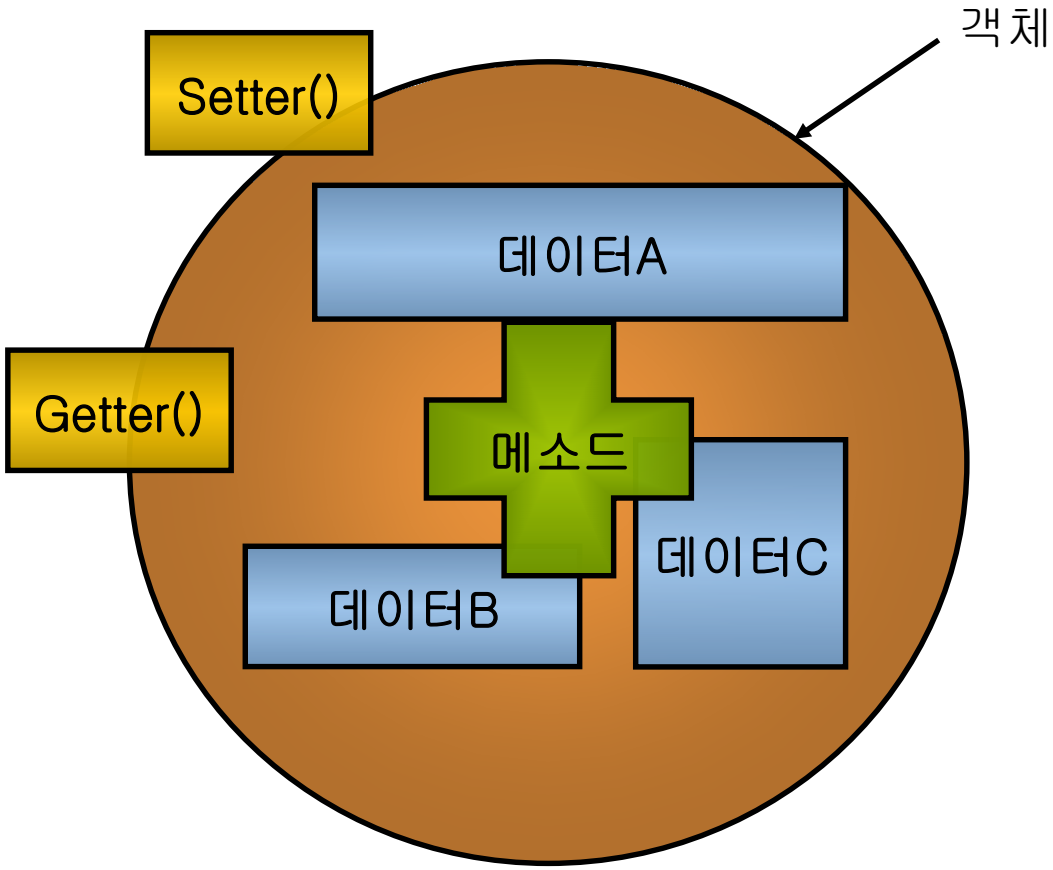
Object Oriented Language

- 재 정의(overriding)
  - 재 정의는 상위 Class로부터 물려 받은 Method를 동일한 형식으로 하위 Class에서 정의함으로써 하위 Class의 상황에 맞추어 다시 구현하는 것을 말함



Object Oriented Language

- 정보 은닉(Information hiding)

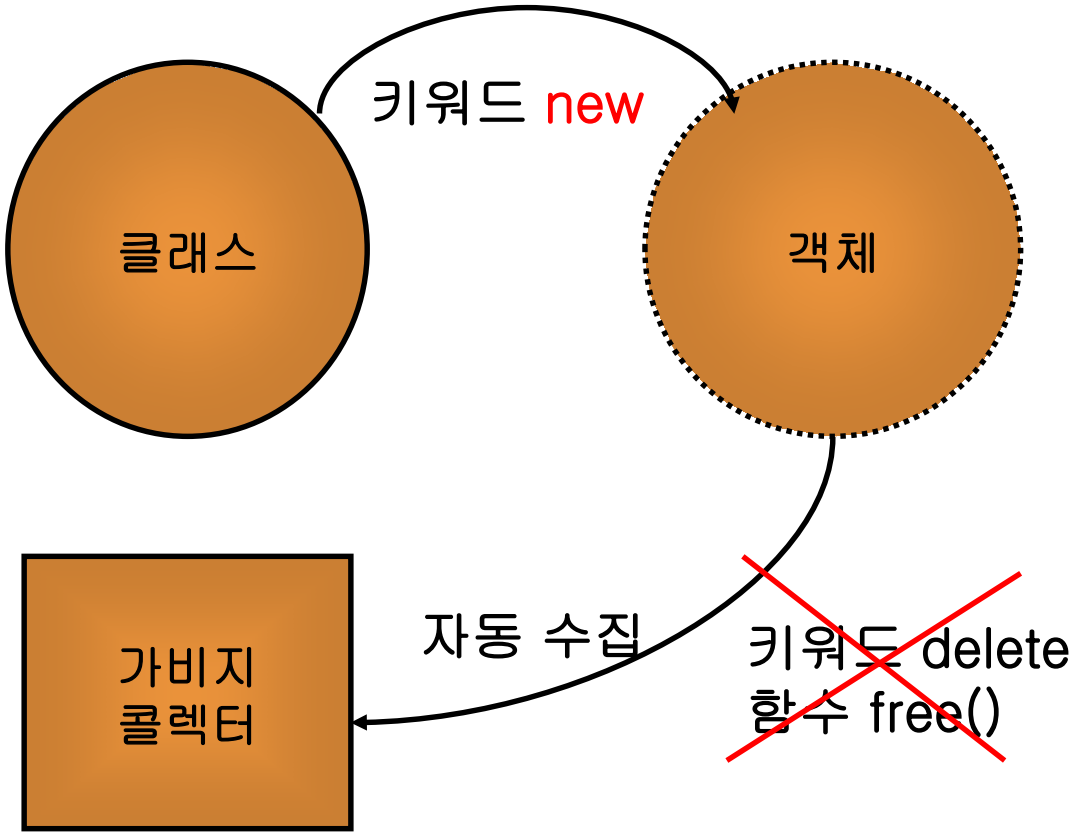


Object Oriented Language

- 메시지 전달(message passing)
  - 객체 지향 관점에서 Program은 Object에 메시지(Message)를 보내는 것으로 실행
  - 점 연산자(.) 활용

Object Oriented Language

- Memory 관리

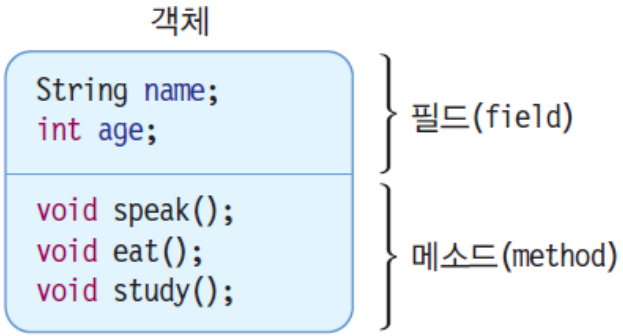


Encapsulation

- Encapsulation은 감기에 걸렸을 때 먹는 캡슐약과 같은 개념
- 캡슐 약에는 많은 성분이 포함되어 있지만, 사용자는 그 캡슐이 단순히 감기를 낮게 해준다고 생각하며 먹고 있으며 약을 먹는 사람은 그 캡슐의 성분을 자세히 알 필요도 없음

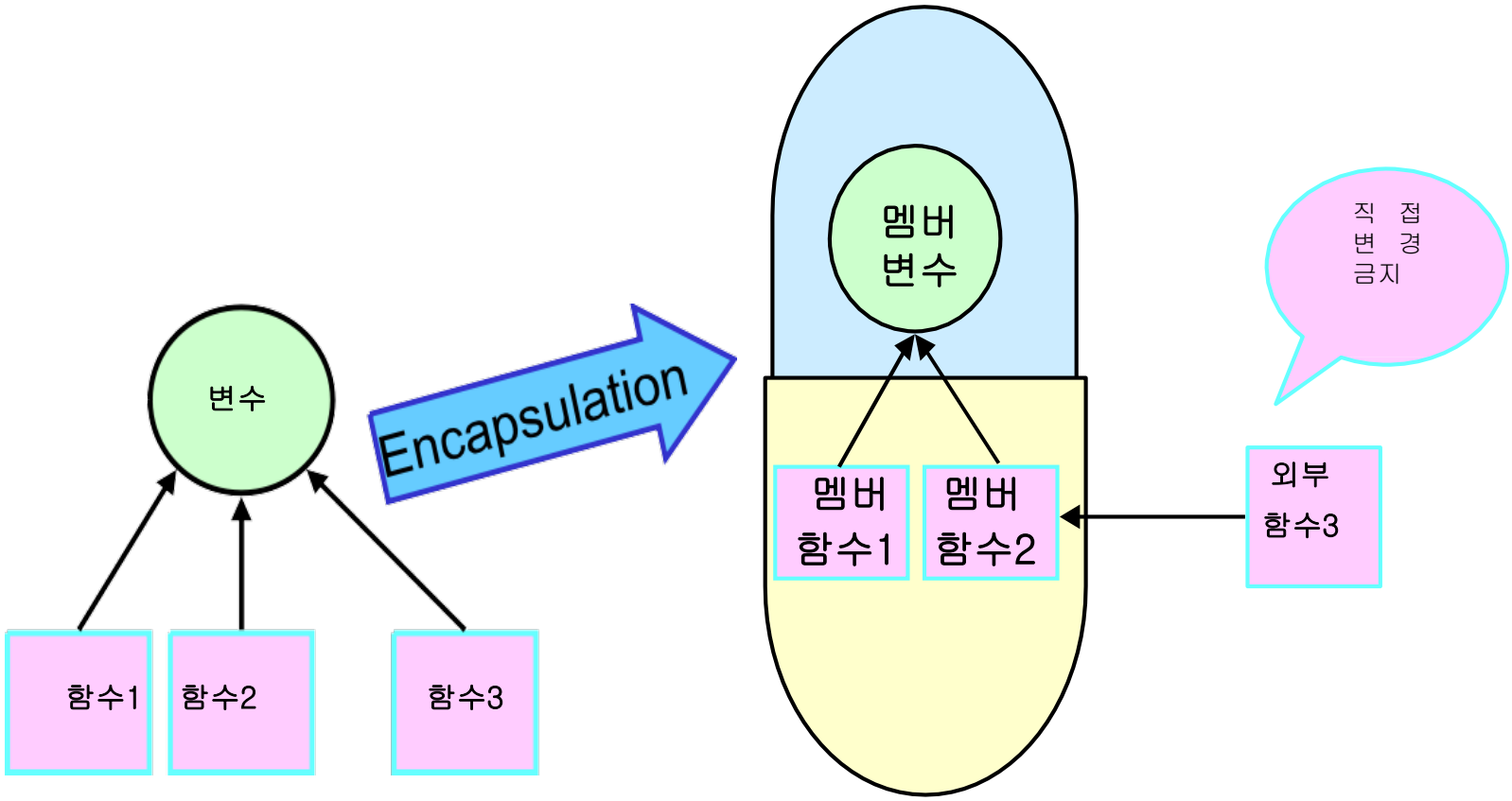
Encapsulation

- Object는 Data(상태 정보)와 Data를 처리하는 Method(서비스 수행)를 가지고 있음
- Object를 사용하는 측면에서는 그 Object의 단순한 기능만 알면 충분히 그 Object를 사용할 수 있으며 Object가 실제 Data를 어떻게 처리하는지는 알 필요가 없다는 것이 캡슐화의 의미
- Object는 내부 구조를 몰라도 단순한 기능과 Interface만 알면 사용이 가능
- Object를 캡슐화하여 What만 보여주고 How는 은폐



자바 객체의 캡슐화

Encapsulation



Encapsulation

- 목적
  - Information hiding
    - 복잡한 내부 구조 숨김
  - Divide and Conquer
    - Program 전체적인 관점에서 문제를 잘게 나누고 해결
  - Programmer 실수 방지
    - 반드시 공개가 필요한 Data 함수로 공개
- 설명
  - Data와 이를 조작하는 Code(Control, Function) 통합
  - Object안의 Code와 Data는 비공개(private)될 수도 있고 공개(public)될 수도 있음

Encapsulation

- 표현된 변수와 메소드(Method)를 하나의 객체에 대한 것으로 묶는 행위
- JAVA에서는 class {. .}와 같이 변수와 메소드(Method)를 하나로 묶음

Inheritance

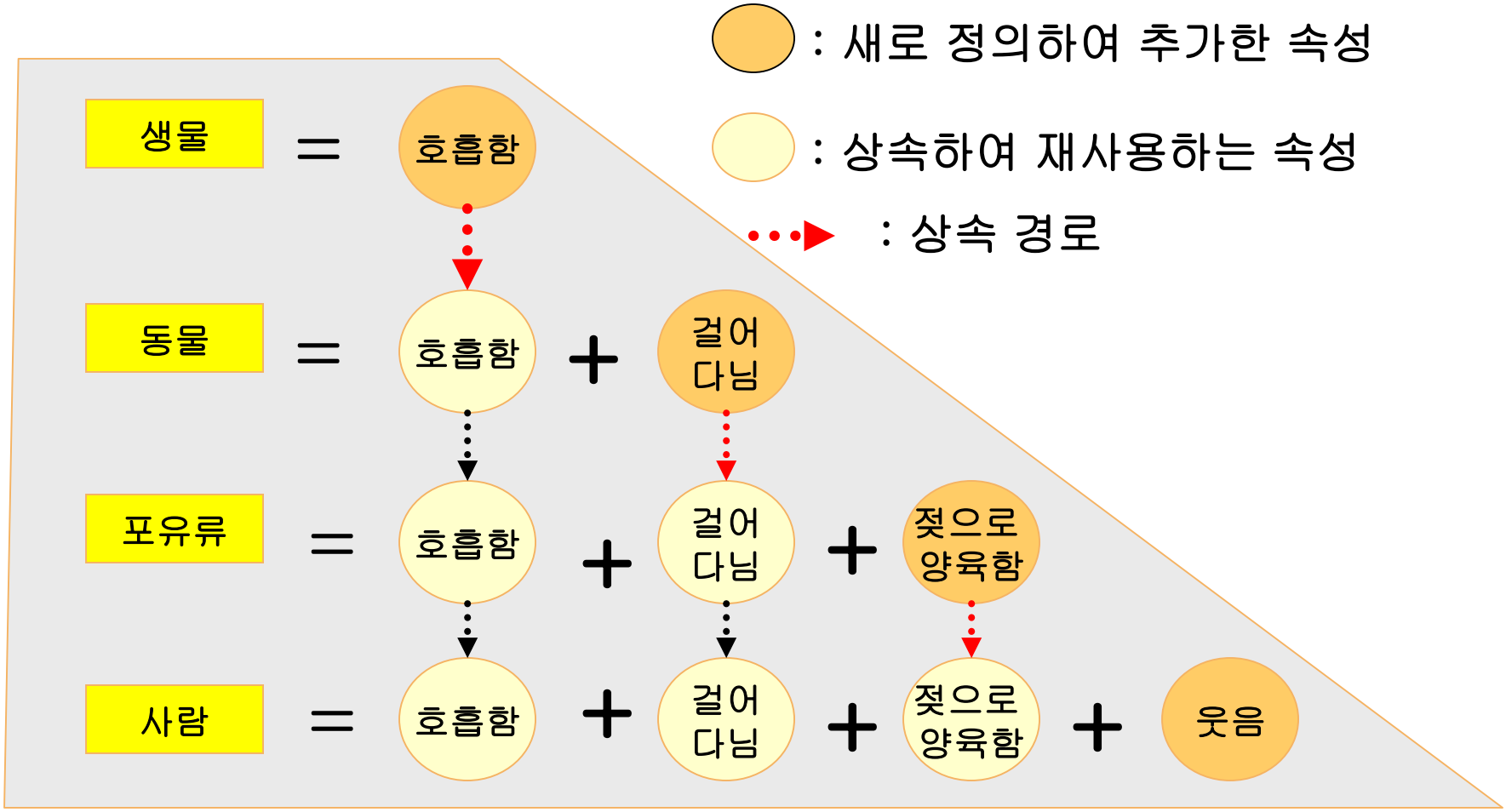
- 상속의 개념
  - Class 계층을 기초로 Class 내의 Data와 Method를 물려받아 새로운 Object를 생성하는 메커니즘
    - 상위 Class의 개념을 하위 Class에 상속
    - 하위 Class에는 상위 Class에 존재하지 않은 새로운 성질을 추가
      - 기존의 자원을 효율적으로 재사용하여 생산성 향상
  - 부모 클래스(Parent Class)/슈퍼 클래스(Super Class)
    - 넓고 일반적인 범위의 Class
  - 자식 클래스(Child Class)/서브 클래스(Sub Class)
    - 세부 분류된 Class

Inheritance

- 기존의 Class에 기초하여 새로운 Class를 정의하는 것
- 앞의 예제에서 생성된 학생 Object들은 같은 Attribute와 Method를 가지고 있음
- 학생 Object와는 비슷하지만 Method가 약간 다른 Object를 생성하려 할 때에는 새로운 Class가 작성되어야 함
- 새로 작성되는 Class는 모든 Attribute와 Method를 상속받고, 더 필요한 Attribute와 Method를 추가하여 새로운 Class를 생성할 수 있음
- Class는 Object의 범주, Object는 Class의 Instance
  - 하나의 Object는 Class의 Instance로서 그 Class의 모든 특성을 이어받음

Inheritance

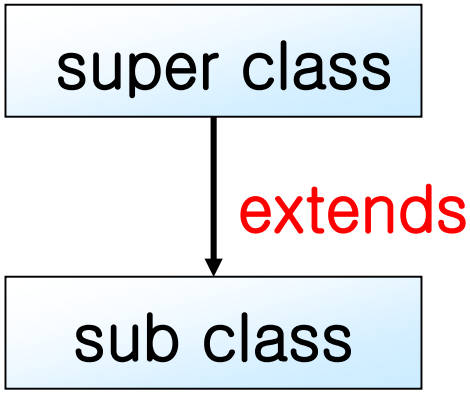
- Class 계층 구조에서의 상속 관계



속성을 상속하여 새로운 클래스를 생성

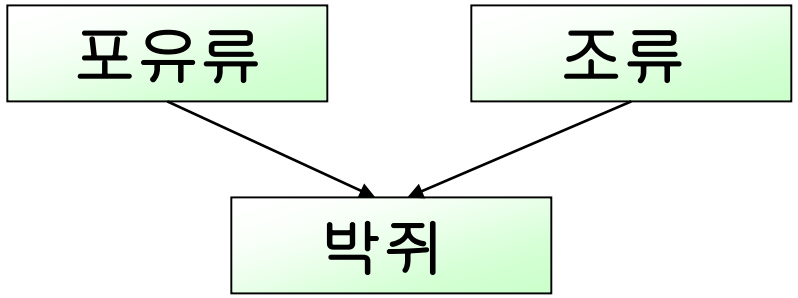
Inheritance

- 상위 클래스(super class)의 모든 Attribute가 하위 클래스(sub class)로 상속 됨
- 일반적으로 계층 관계에 있는 Class간에는 상속 관계가 성립, 상위 Class의 Attribute와 Method가 모두 하위 Class에 상속됨
- Class의 Instance인 Object를 생성하지 못하고 하위 Class에 상속만 되는 Class를 Abstract Class라 함
- JAVA에서는 상속 관계를 표현하기 위해 extends라는 키워드를 사용함



Multiple Inheritance

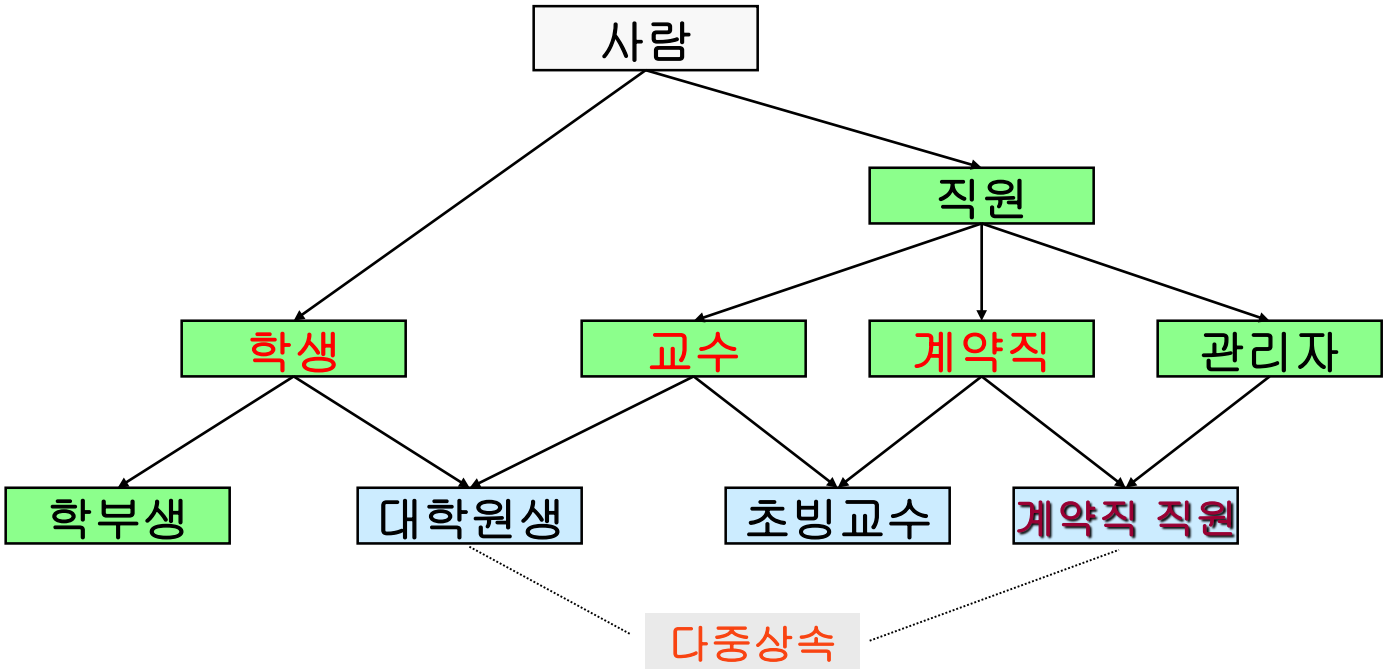
- 일반적인 Class 계층 관계에서는 하나의 상위 Class로부터 Attribute와 Method를 상속 받는 것이 일반적
- 한 Class가 2개 이상의 상위 Class로부터 Attribute를 상속 받을 수 있음 (JAVA는 안됨)
- 예) 박쥐라는 동물은 조류의 특성과 포유류의 특성을 모두 상속 받음



다중 상속의 예

Multiple Inheritance

- 상속 계층에서 하나의 Class가 여러 개의 Supper Class를 가지는 상속
  - 주어진 문제 영역에서 다중 상속이 필요한 경우 그대로 표현할 수 있으므로 편리
- 예) 대학 인사관리시스템

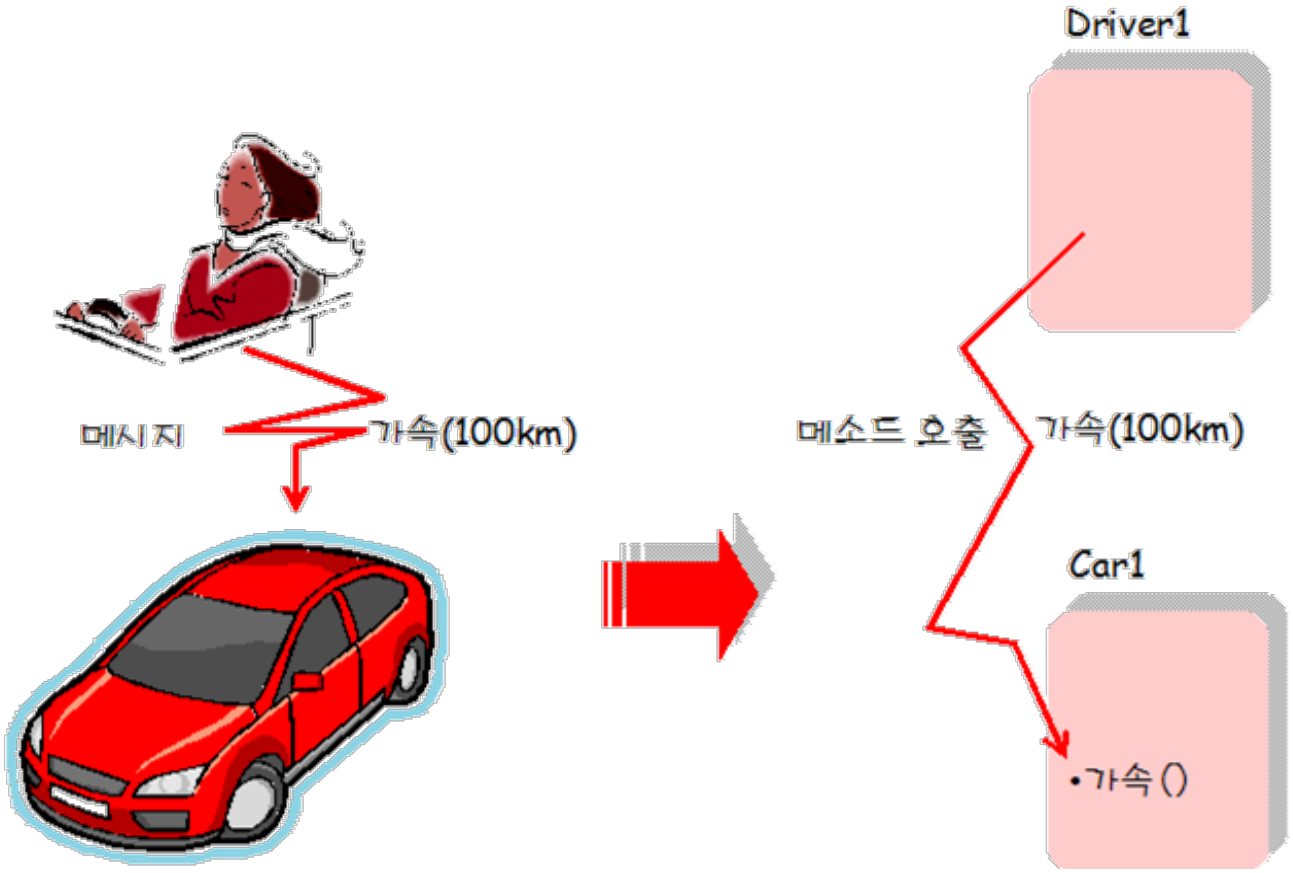


Multiple Inheritance

- 일반적으로 Object-Oriented Programming에서는 다중 상속이 지원됨
- 다중 상속된 하위 클래스는 상위 클래스들이 가진 모든 속성과 메소드를 상속 받음
- JAVA 언어의 설계자가 다중 상속이 개념적으로 혼란을 줄 수 있으므로, 명칭 충돌(name conflict)과 같은 문제가 발생하므로, JAVA는 다중 상속을 지원하지 않도록 설계 하였음
- JAVA 클래스는 많아야 하나의 상위 클래스를 가질 수 있음
- JAVA Programming에서는 사용자가 정의한 클래스는 JAVA System에서 정의한 클래스로부터 상속받아야 하므로, 다중 상속이 필요한 경우가 많이 발생함
- 이를 위해서 JAVA는 비록 명시적인 다중 상속을 지원하지는 않지만, implements라는 Keyword를 사용하여 여러 Interface를 구현함

Message

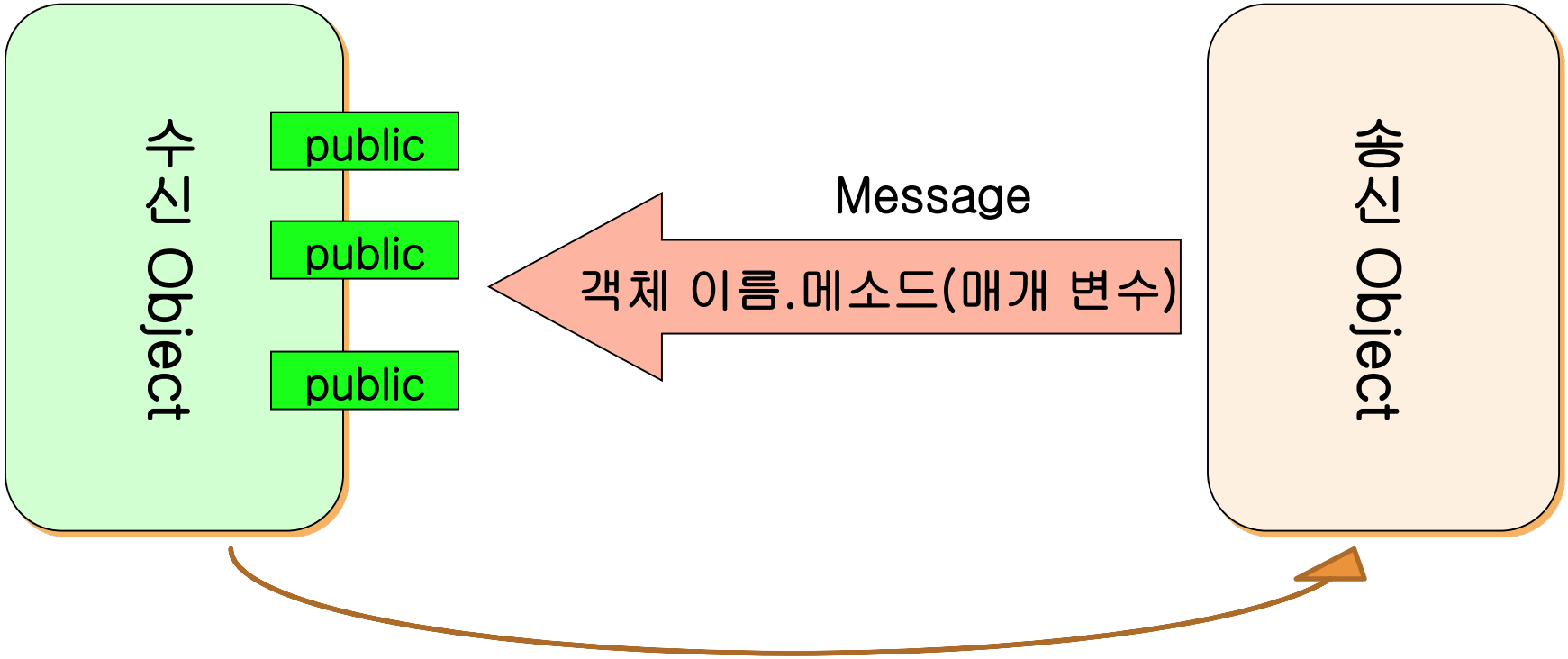
- Software Object는 Message를 통해 다른 Software Object와 통신하고 서로 상호 작용함



Message

- Message
  - Object에게 일을 하게 하는 행위
  - Object간의 통신 수단
  - Program에서 생성된 Object들은 이러한 Message를 주고받음으로 서 일을 수행
  - Programmer는 사용하고자 하는 객체를 정의한 다음 이러한 객체들이 어떤 일을 수행해야 하는지 Message로 작성
- Message의 3요소
  - Message를 받을 Object의 이름(주소)
  - 송신 Object가 실행을 원하는 수신 Object의 Method 이름
  - 실행을 원하는 Method에 전달할 매개 변수

Message



Message를 전달 받은 Object는 Message의 내용을 분석하여 Message에 지정된 Method 를 수행하여 결과를 반환

Message

- Message Passing

```
class 자동차 {
   String 용도, 변속타입, 색상;
   int 엔진출력;

   전진하기(int speed);
   후진하기(int speed);
   방향틀기(int angle);
   정지하기();
   전조등켜기(boolean b);
   경적소리내기(int level);
}
```

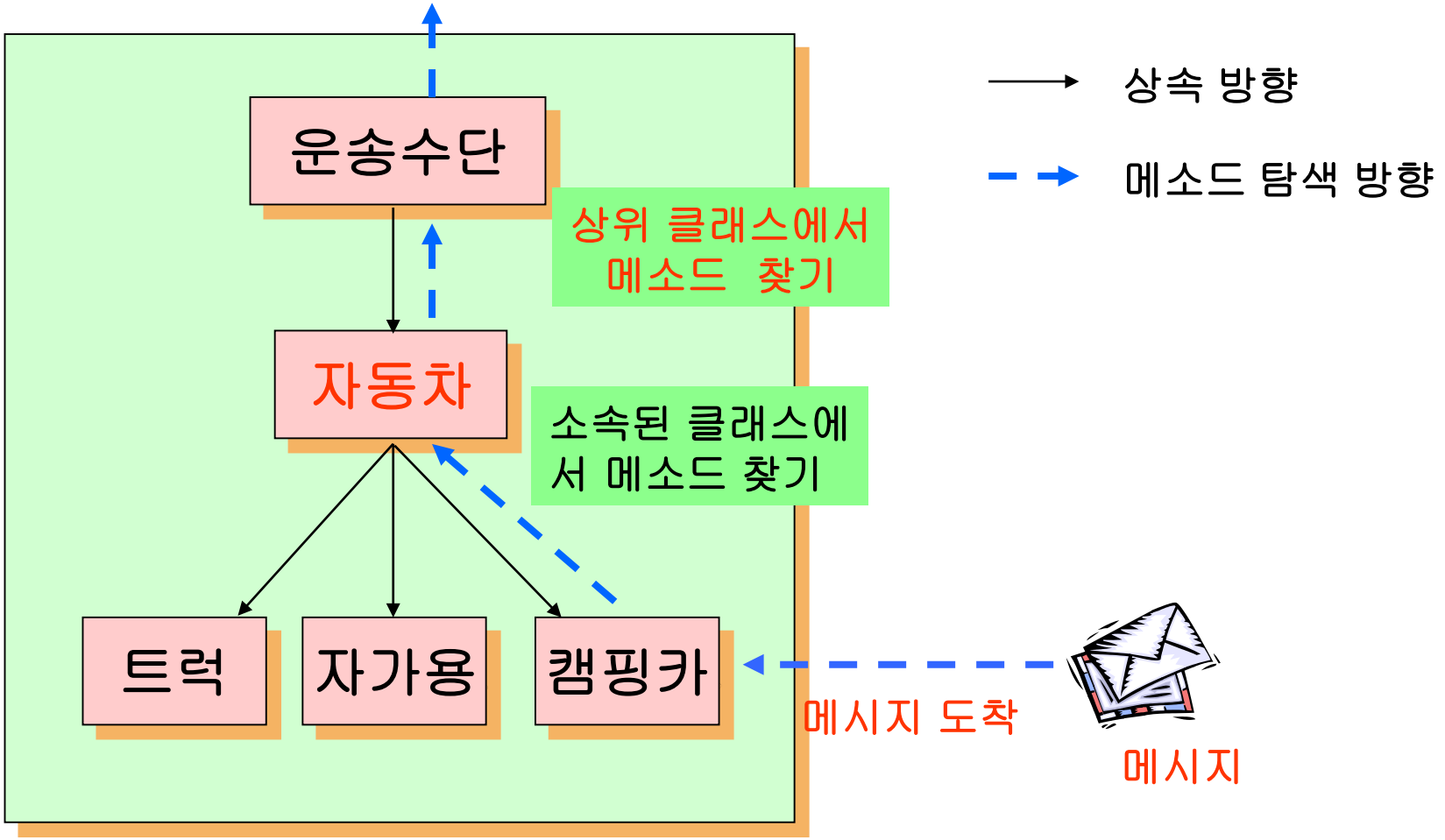
클래스 자동차의 구조

```
class 자동차운전 {

public static void main (String args[]) {
   자동차 세피아;

   소나타 = new 자동차(자가용, 1500, 수동,
                검정);

   소나타.전조등켜기(true);
   소나타.전진하기(30);
   소나타.정지하기();
}
```

소나타 객체의 메소드 호출

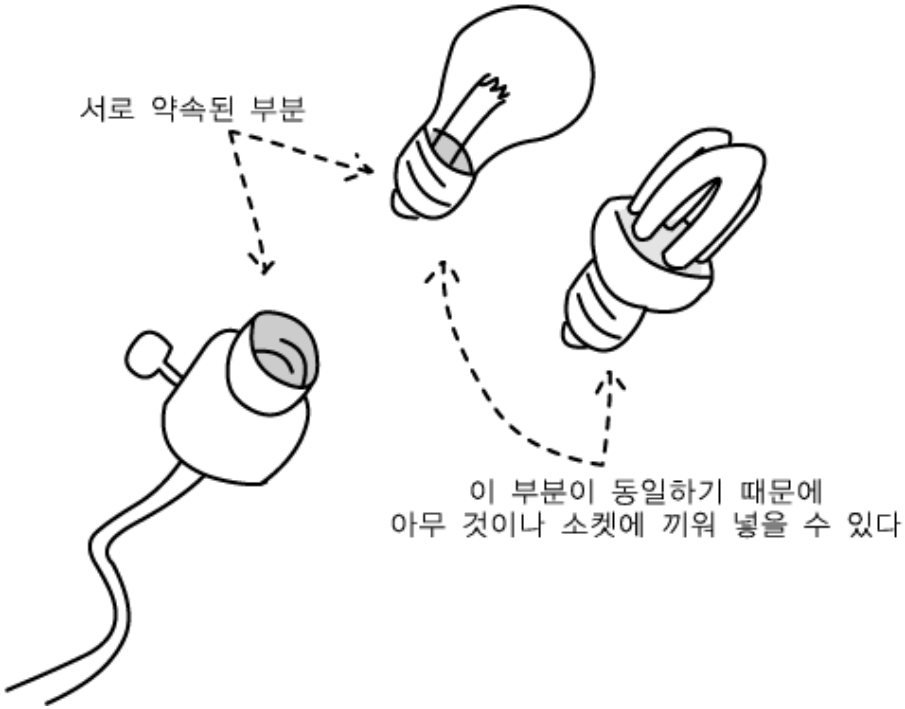
Message

- Message Passing



Polymorphism

- 다형성이란 서로 다른 Object를 동일한 방식으로 명령을 내릴 수 있는 성질을 말함
- 이 때 서로 다른 Object들은 같은 명령을 받지만 제각기 다른 방식으로 명령을 수행할 수 있음
  - 예) 백열등과 삼파장 램프는 동일한 소켓에 끼워서 사용할 수 있음



Polymorphism

- 복수의 Class가 하나의 Message에 대해 각 Class가 가지고 있는 고유한 방법으로 응답할 수 있는 능력
- 별개로 정의된 Class들이 같은 이름의 함수를 별도로 가지고 있어 하나의 Message를 각기 다른 방법으로 해석하여 필요한 함수를 수행할 수 있는 것을 말함

- Virtual Function을 사용하여 구현
  - Class를 정의할 때, "virtual"이라는 키워드를 사용하여 정의되는 멤버 함수
  - 가상 함수는 일반 함수와 달리 Program이 Compile될 때 Binding(정적 바인딩)이 이루어지는 것이 아니라, Program이 실행될 때 Binding(동적 바인딩)이 이루어짐

Polymorphism

- Compile time Polymorphism
  - 함수(Function), 연산자(Operator)의 Overloading
- Run time Polymorphism
  - 가상함수를 이용한 함수의 오버라이딩(Overriding)

Polymorphism

- Overloading(다중정의)
  - 같은 이름의 함수/연산자를 여러 기능으로 사용하는 것
  - JAVA에서는 한 함수가 다른 전달인자(parameter)형들을 위해서 여러 번 정의될 수 있음

```
class Overload {
        static void find()  {
            System.out.println("No argument");
         }

        static void find(int x) {
            System.out.println("int arg =" + x);
        }
}
```

Polymorphism

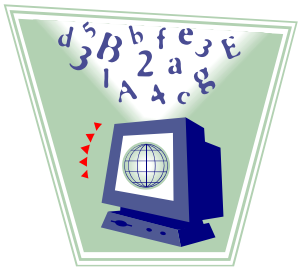
- Overloading
  - 한 Class 안에는 같은 이름의 Method가 다수 존재함
- 규칙
  - 한 Class 안에서 같은 이름의 Method
  - 중복정의하고 자 하는 Method
    - 반드시 서로 다른 형의 매개 변수를 가지고 있거나, 매개 변수의 개수가 달라야 함
- 특징
  - 중복 정의된 Method는 Compile에 의해 구별
  - 매개 변수의 개수가 같고 매개 변수의 형이 다른 경우
    - 동일한 형의 매개 변수를 갖는 Method를 찾음
    - 만약 찾지 못하면 기본적인 형 변환에 의해 실 매개 변수의 형을 형식 매개 변수의 형으로 바꿀 수 있는 Method를 찾음

Polymorphism

- 목적
  - 같은 이름의 Method로 하여금 서로 다른 기능을 부여하고자 할 때 주로 사용
  - 예) printOut( )

문자열 출력 시	음성 출력 시

public void printOut(String s) {
  문자를 출력하기 위한 코드 ;
}
	public void printOut(Audio a) {
	  음성을 출력하기 위한 코드 ;
	}

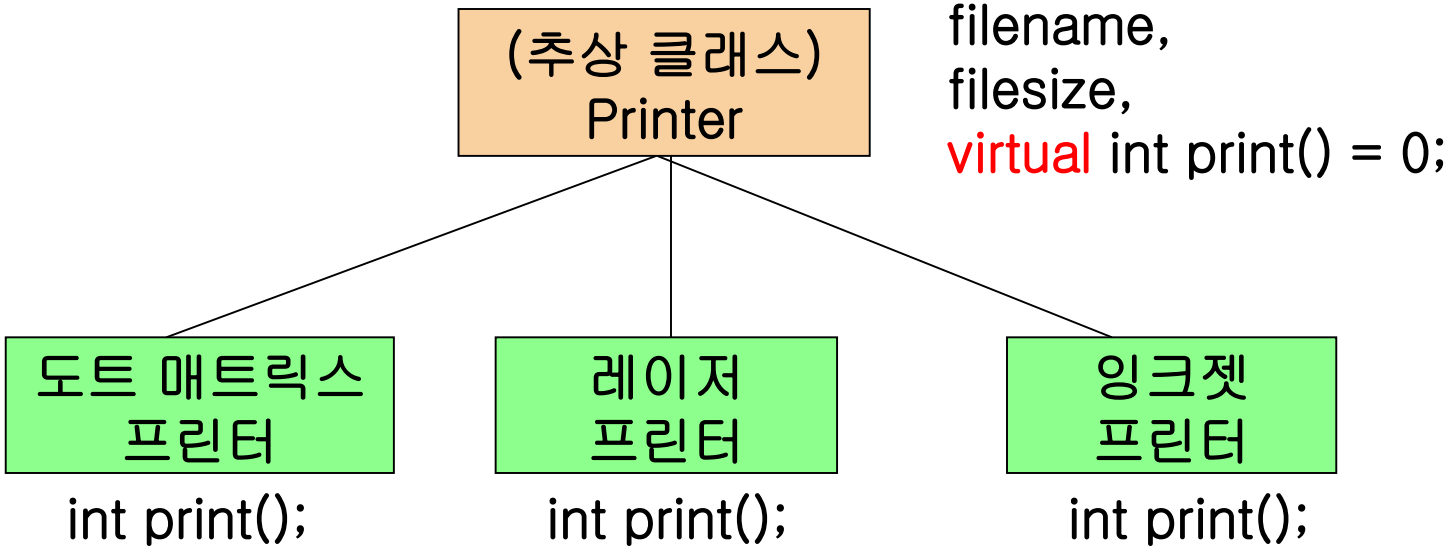


Polymorphism

- Printer Software
  - 모든 종류의 프린터가 공통으로 가지는 특성을 추상 클래스에 정의
    - 변수 : 프린터의 상태, 속도
    - 함수 : print();
    - 프린트마다 인쇄하는 방법이 다르므로 이 추상 클래스 안에서는 print()라는 함수를 완전히 구현할 수 없음
    - 실제 구현은 여러 서브 클래스에서 각 프린터 종류에 맞게 하면 됨

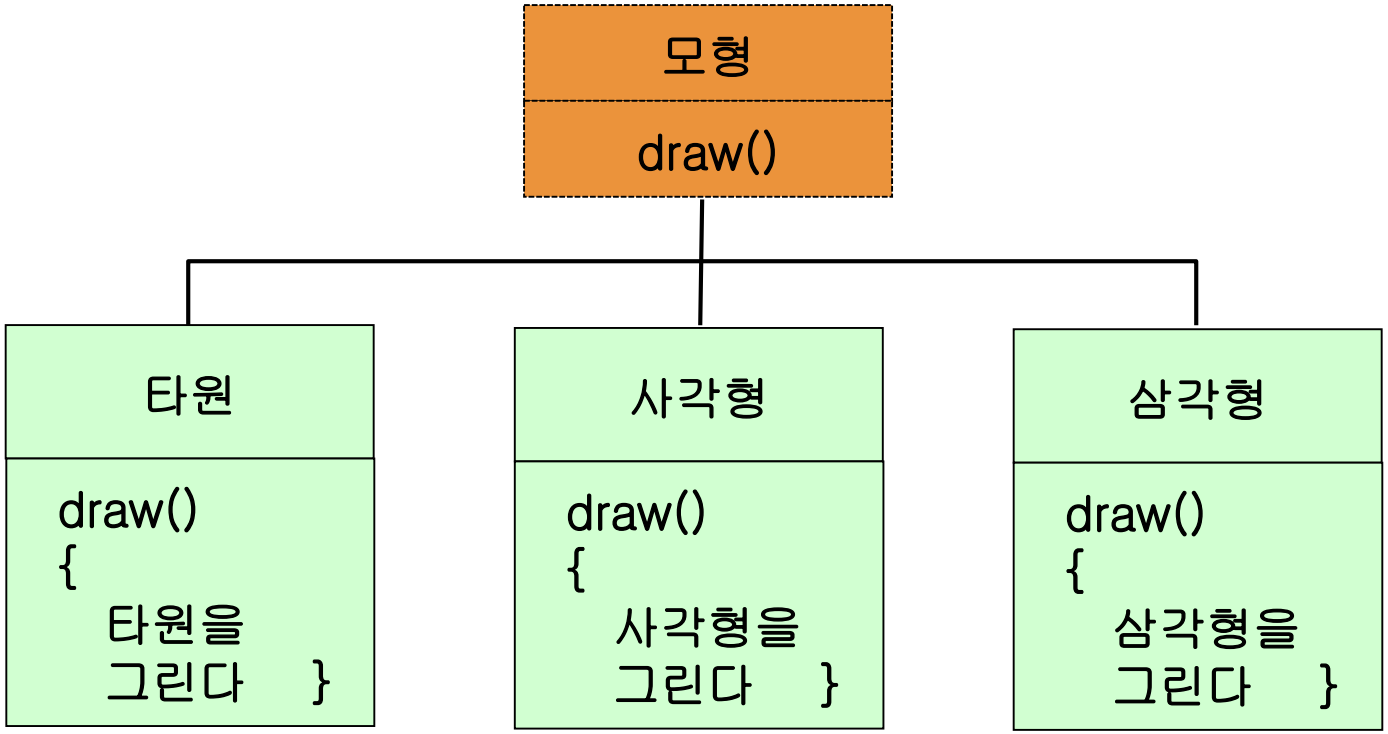
Polymorphism

- 추상 클래스 'Printer'
  - 각 파일의 종류별로 출력하는 알고리즘이 다르나, 함수의 이름은 모두 print()로 정의함
  - Polymorphic하게 print()함수가 정의 되었음



Polymorphism

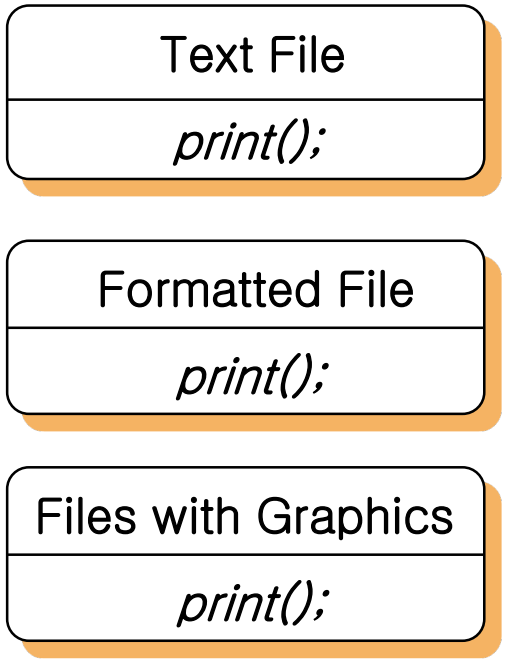
- “one interface, multiple implementation”
- 하나의 인터페이스를 사용하여 다양한 구현 방법을 제공

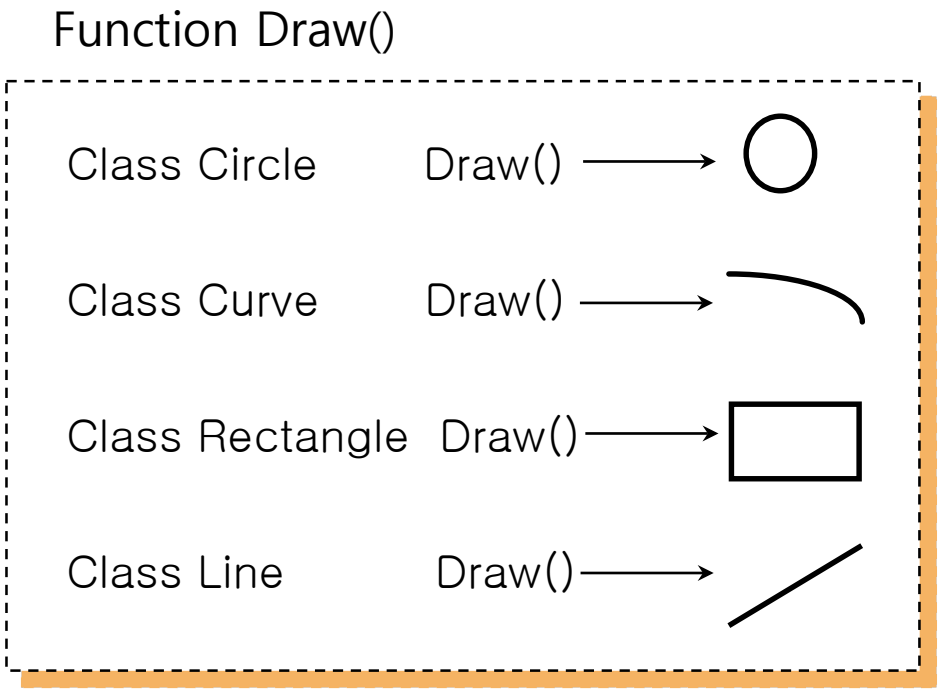


Polymorphism

- 여러 개의 Class가 하나의 Message에 각 Class가 가지고 있는 고유한 방법으로 응답할 수 있는 능력
- 장점
  - Module 및 재 사용성이 향상, 유지보수가 용이

"print" **함수의 복수 정의**





Polymorphism

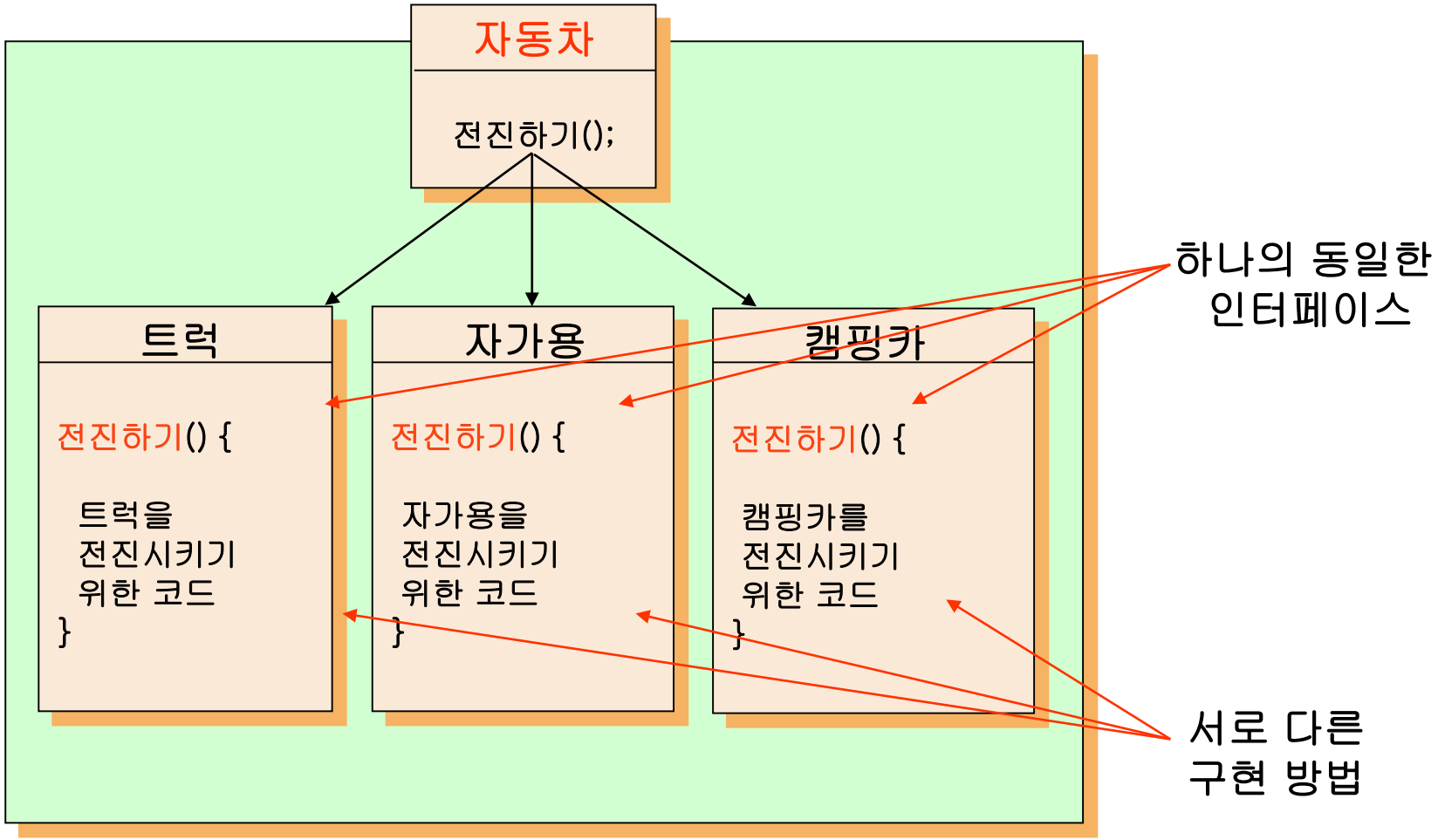
- Overriding(오버라이딩, 재정의)
  - 파생 Class를 정의할 때, 기존의 기반 Class에 포함되어 있는 Member 함수들 중에서 하나를 선택하여 새로운 내용으로 정의할 수 있다는 것을 의미
  - 함수 오버라이딩은 Class를 사용한 객체 지향 프로그래밍에 있어서, 각 Module을 캡슐화하면서 동일한 함수에 새로운 기능을 추가하거나 프로그램 특성에 맞게 함수를 재구성할 때 매우 유용
  - 일반적으로 파생된 클래스에서 함수를 오버라이딩하기 위해서는 반드시 오버라이딩되는 함수와 같은 수 만큼의 매개 변수를 같은 순서로 같은 데이터 형을 사용하여 정의해야 함

Polymorphism

- 매개변수(Parameter) 수가 다른 함수의 Overloading
  - 함수 Overloading시 항상 매개 변수의 개수가 같아야 하는 것은 아님
  - 필요에 따라 매개 변수의 개수가 같지 않게 정의할 수 있음

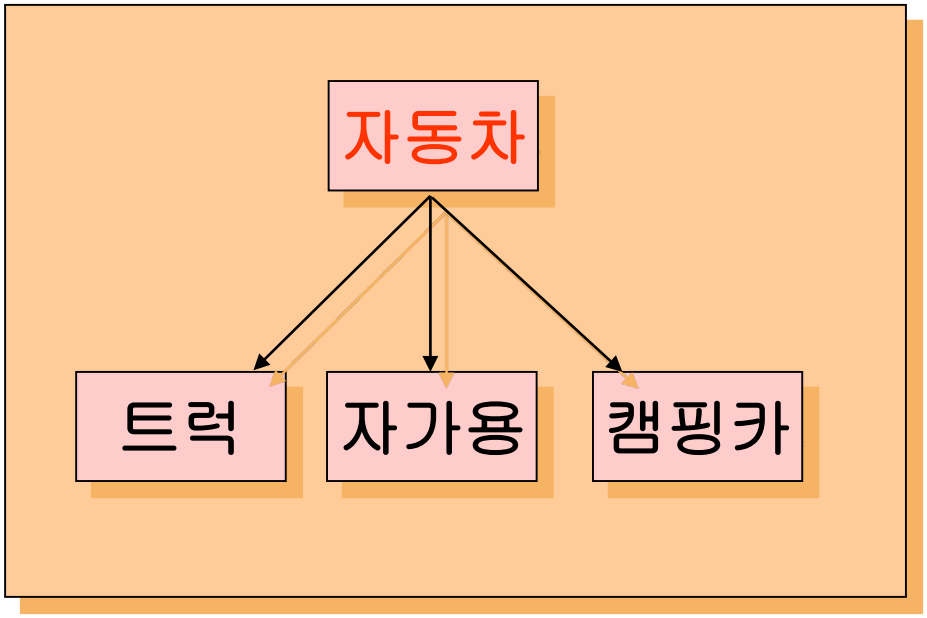
Polymorphism

- 같은 이름의 Method를 허용함



Polymorphism

- 다형성과 유효 객체 할당 방법



○

```
자동차 홍길동차;
홍길동차 = new 자가용();
```

상위 클래스(자동차) 타입의 객체변수(홍길동차)에 하위 클래스(자가용) 객체 할당은 가능

×

```
자가용 홍길동차;
홍길동차 = new 자동차();
```

하위 클래스(자가용) 타입의 객체변수(홍길동차)에 상위 클래스(자동차) 객체 할당은 불가