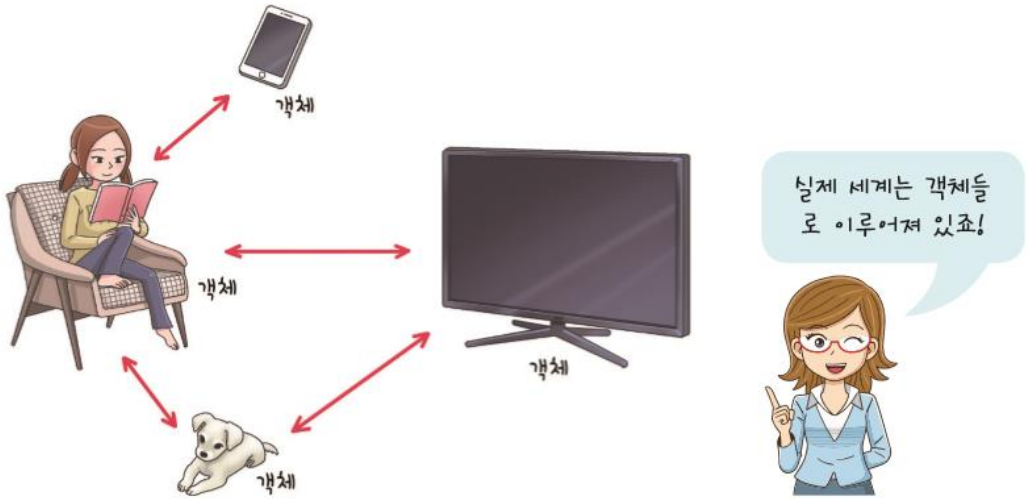
Object Oriented Programming

- 객체(object)는 함수와 변수를 하나의 단위로 묶을 수 있는 방법이다. 이러한 프로그래밍 방식을 객체지향(object-oriented)이라고 한다.



- 객체란?
  - 객체는 하나의 물건이라고 생각하면 된다. 객체는 속성(attribute)과 동작(action)을 가지고 있다.

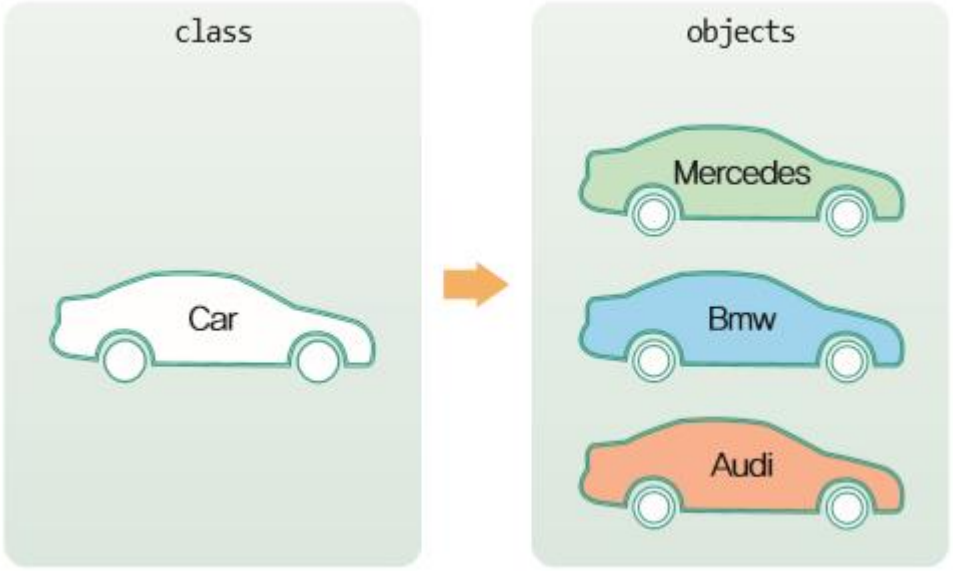
속성
메이커
모델
색상
연식
가격

동작
주행하기
방향바꾸기
주차하기

- 클래스란?
  - 객체에 대한 설계도를 클래스(class)라고 한다.
  - 클래스로부터 만들어지는 각각의 객체를 그 클래스의 인스턴스(instance)라고 한다.

- 객체 생성하기

1. 객체의 설계도를 작성하여야 한다.
2. 클래스로부터 객체를 생성하여야 한다.



```
class Car:
    def __init__(self, speed, color, model):
        self.speed = speed
        self.color = color
        self.model = model

    def drive(self):
        self.speed = 60

myCar = Car(0, "blue", "E-class")

print("자동차 객체를 생성하였습니다.")
print("자동차의 속도는", myCar.speed)
print("자동차의 색상은", myCar.color)
print("자동차의 모델은", myCar.model)
print("자동차를 주행합니다.")
myCar.drive()
print("자동차의 속도는", myCar.speed)
```

```
자동차 객체를 생성하였습니다.
자동차의 속도는 0
자동차의 색상은 blue
자동차의 모델은 E-class
자동차를 주행합니다.
자동차의 속도는 60
```

- 클래스를 만들고, 객체를 생성해보자

In [4]:
```
class Person:
    pass
```

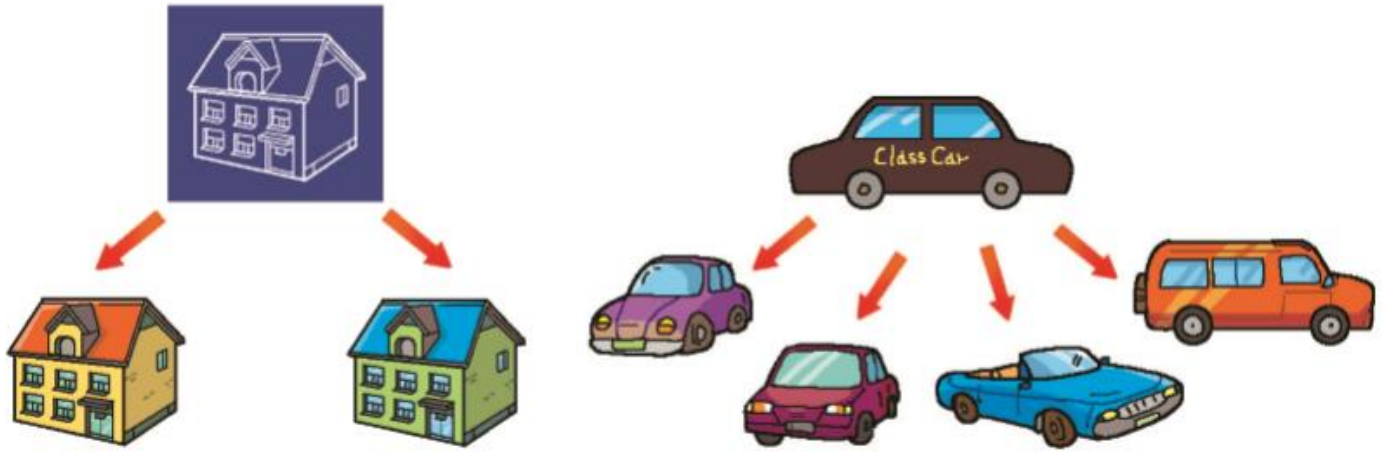
In [2]:
```
p = Person()
print(p)
```

```
<__main__.Person object at 0x0000014F665B07F0>
```

- 클래스 내 함수(메소드)에서 사용되는 self
  - say_hi 메소드는 어떤값(매개변수)도 받지 않음
  - say_hi 메소드는 함수정의에 self 를 매개변수 가지고 있음

```
In [7]: class Person:
            def say_hi(self):
                print('hello~')

In [8]: p = Person()

In [9]: p.say_hi()
hello~
```

- 하나의 클래스로 객체는 많이 만들 수 있다.
  - 우리는 하나의 클래스로 여러 개의 객체를 생성할 수 있다



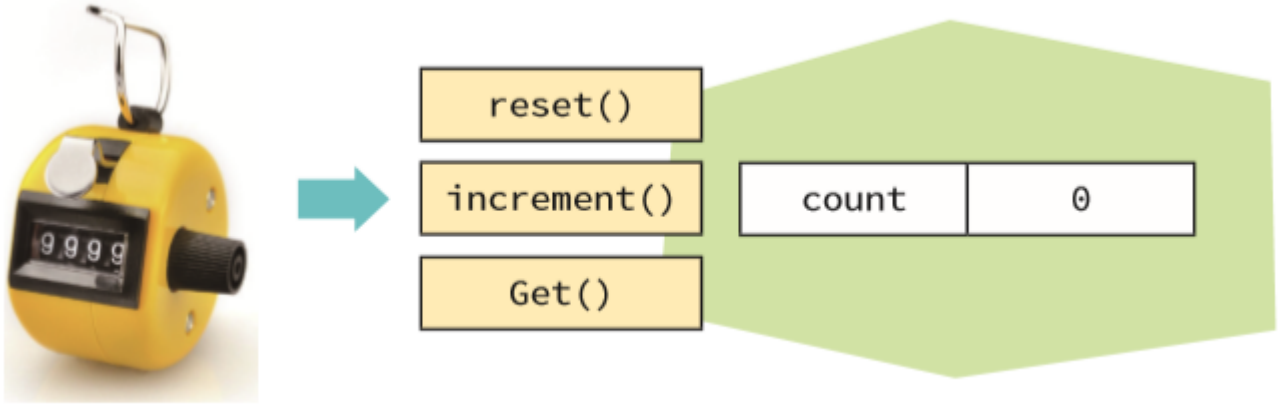
- 클래스 작성하기

전체적인 구조

```
class 클래스 이름:
    def 메소드1(self, ...):
        ...
    def 메소드2(self, ...):
        ...
```

메소드를 정의한다.

- 클래스의 예
  - Counter 클래스를 작성하여 보자. Counter 클래스는 기계식 계수기를 나타내며 경기장이나 콘서트에 입장하는 관객 수를 세기 위하여 사용할 수 있다.



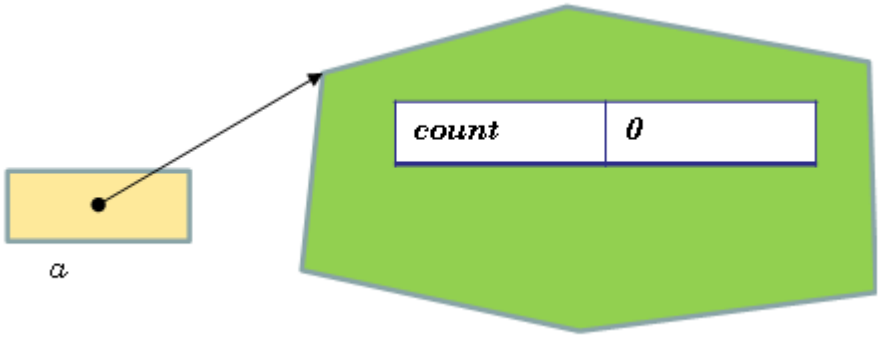
- Count 클래스

```
class Counter:
    def reset(self):
        self.count = 0
    def increment(self):
        self.count += 1
    def get(self):
        return self.count
```

- 객체 생성

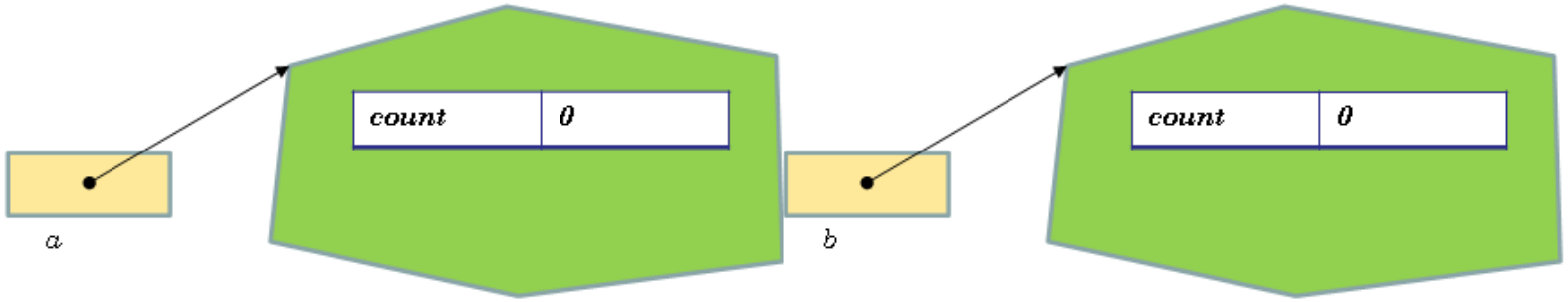
```
a = Counter()

a.reset()
a.increment()
print("카운터 a의 값은", a.get())
```

```
카운터 a의 값은 1
```

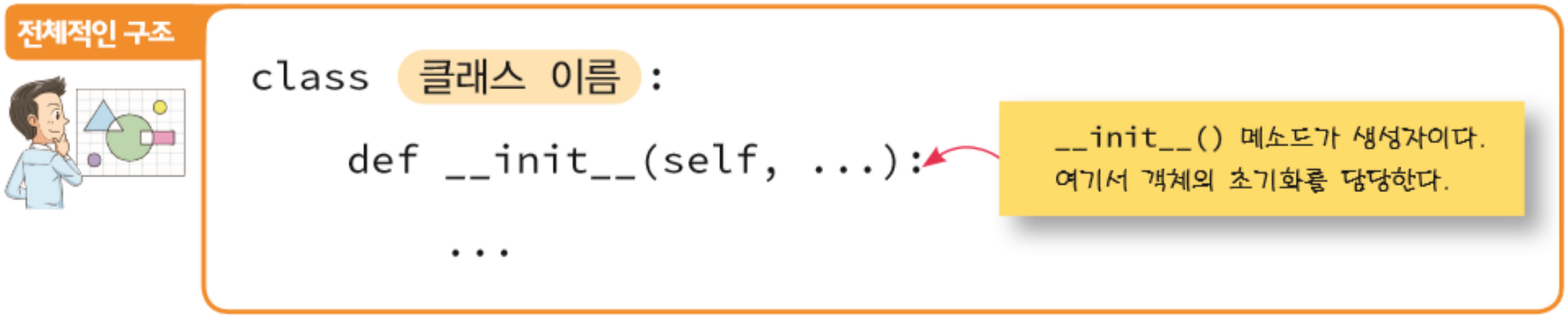


- 객체 2개 생성하기

```
a = Counter()
b = Counter()

a.reset()
b.reset()
```



- __init__(self)의 예




```
class Counter:
    def __init__(self) :
        self.count = 0
    def reset(self) :
        self.count = 0
    def increment(self):
        self.count += 1
    def get(self):
        return self.count
```

- init 메소드는 클래스가 인스턴스화 될 때 호출
- 객체가 생성시 초기화 명령이 필요할 때 유용하게 사용
- init 의 앞과 뒤에 있는 밑줄은 두 번씩 입력

In [10]:
```
class Person:
    def __init__(self, name):
        self.name = name
    def say_hi(self):
        print('hello~', self.name)
```

In [12]:
```
p = Person('길동')
p.say_hi()
```

```
hello~ 길동
```

- 클래스 변수
  - 공유됨
  - 모든 인스턴스들이 접근할 수 있음
  - 한 개만 존재
  - 객체가 값을 변경하면 다른 인스턴스에 적용됨

- 객체변수
  - 개별 객체 / 인스턴스에 속함 (독립)
  - 다른 인스턴스들이 접근할 수 없음
  - self 에 연결(self.name)

- 클래스 함수 만드는법

- @classmethod

        def how_many(cls):

```
class Robot:
    '''로봇 클래스'''
    population = 0                          #클래스 변수

    def __init__(self, name):
        '''먼저 시작되는 메소드'''
        self.name = name                    #객체변수
        print('시작합니다.', self.name)
        Robot.population += 1

    def die(self):
        '''로봇 파괴'''
        print(self.name, '파괴되었습니다.')

        Robot.population -= 1
        if Robot.population == 0:
            print(self.name, '마지막 로봇입니다.')
        else:
            print(Robot.population, '남았습니다.')
```

```
def say_hi(self):
    '''인사'''
    print(self.name, '반갑습니다.')

@classmethod
def how_many(cls):
    print(cls.population, '가지고 있습니다.')
```

```
In [44]: droid1 = Robot('길동1')
```

```
시작합니다. 길동1
```

```
In [45]: droid1.say_hi()
```

```
길동1 반갑습니다.
```

```
In [46]: Robot.how_many()
```

```
1 가지고 있습니다.
```

```
In [47]: droid2 = Robot('길동2')
```

```
시작합니다. 길동2
```

```
In [47]: droid2 = Robot('길동2')
```

```
시작합니다. 길동2
```

```
In [48]: droid2.say_hi()
```

```
길동2 반갑습니다.
```

```
In [49]: Robot.how_many()
```

```
2 가지고 있습니다.
```

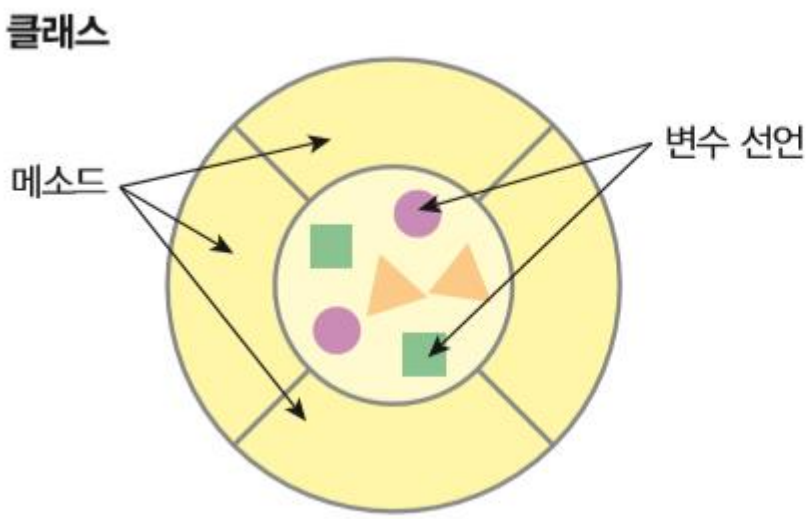
```
In [50]: droid2.die()
```

```
길동2 파괴되었습니다.
1 남았습니다.
```

```
In [51]: Robot.how_many()
```

```
1 가지고 있습니다.
```

- 정보 은닉
  - 구현의 세부 사항을 클래스 안에 감추는 것



- 파이썬에서 클래스 private 멤버로 정의 하려면 변수 이름 앞에 __을 붙이면 된다.
- 이 변수는 클래스 내부에서만 접근 될 수 있다.

```
class Student:
        def __init__(self, name=None, age=0):
                self.__name = name
                self.__age = age

obj=Student()
print(obj.__age)
```

```
...
AttributeError: 'Student' object has no attribute '__age'
```

- 접근자와 설정자
  - 하나는 인스턴스 변수값을 반환하는 **접근자(getters)**이고 또 하나는 인스턴스 변수값을 설정하는 **설정자(setters)**이다.

```
class Student:
        def __init__(self, name=None, age=0):
                self.__name = name
                self.__age = age

        def getAge(self):               # 접근자
                return self.__age

        def getName(self):
                return self.__name

        def setAge(self, age):          # 설정자
                self.__age=age

        def setName(self, name):
                self.__name=name
```

```
In [60]: s = Student()

In [61]: print(s.__age)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-61-37cec98c33ac> in <module>
----> 1 print(s.__age)

AttributeError: 'Student' object has no attribute '__age'
```

```
In [71]: s.setAge(20)

In [76]: print(s.getAge())
```

```
20
```

OOP 실습 예제

- 원을 클래스로 표현하기
  - 원을 클래스도 표시해보자. 원은 반지름(radius)을 가지고 있다. 원의 넓이와 둘레를 계산하는 메소드도 정의해보자. 설정자와 접근자 메소드도 작성한다.

```
원의 반지름= 10
원의 넓이= 314.0
원의 둘레= 62.80000000000004
```

```
In [78]: c1=Circle(10)
         print("원의 반지름=", c1.getRadius())
         print("원의 넓이=", c1.calcArea())
         print("원의 둘레=", c1.calcCircum())
```

```
class Circle:
        def __init__(self, radius=1.0):
                self.__radius = radius

        def setRadius(self, r):
                self.__radius = r

        def getRadius(self):
                return self.__radius

        def calcArea(self):
                area = 3.14*self.__radius*self.__radius
                return area
        def calcCircum(self):
                circumference = 2.0*3.14*self.__radius
                return circumference
```

In [78]:
```
c1=Circle(10)
print("원의 반지름=", c1.getRadius())
print("원의 넓이=", c1.calcArea())
print("원의 둘레=", c1.calcCircum())
```

```
원의 반지름= 10
원의 넓이= 314.0
원의 둘레= 62.800000000000004
```

OOP 실습예제

- 은행 계좌
  - 우리는 은행 계좌에 돈을 저금할 수 있고 인출할 수도 있다. 은행 계좌를 클래스로 모델링하여 보자. 은행 계좌는 현재 잔액(balance)만을 인스턴스 변수로 가진다. __init__와 인출 메소드 withdraw()와 저축 메소드 deposit() 만을 가정하자.

```
통장에서  100 가 출금되었음
통장에  10 가 입금되었음
```

```
In [81]: a = BankAccount()

In [82]: a.deposit(100)
         통장에서  100 가 출금되었음
Out[82]: 100

In [83]: a.withdraw(10)
         통장에  10 가 입금되었음
Out[83]: 90
```

```
class BankAccount:
    def __init__(self):
        self.__balance = 0

    def withdraw(self, amount):
        self.__balance -= amount
        print("통장에 ", amount, "가 입금되었음")
        return self.__balance

    def deposit(self, amount):
        self.__balance += amount
        print("통장에서 ", amount, "가 출금되었음")
        return self.__balance
```

- 객체를 함수로 전달할 때
  - 우리가 작성한 객체가 전달되면 함수가 객체를 변경할 수 있다.

```
# 사각형을 클래스로 정의한다.
class Rectangle:
        def __init__(self, side=0):
                self.side = side

        def getArea(self):
                return self.side*self.side

# 사각형 객체와 반복횟수를 받아서 변을 증가시키면서 면적을 출력한다.
def printAreas(r, n):
        while n >= 1:
                print(r.side, "\t", r.getArea())
                r.side = r.side + 1
                n = n - 1
```

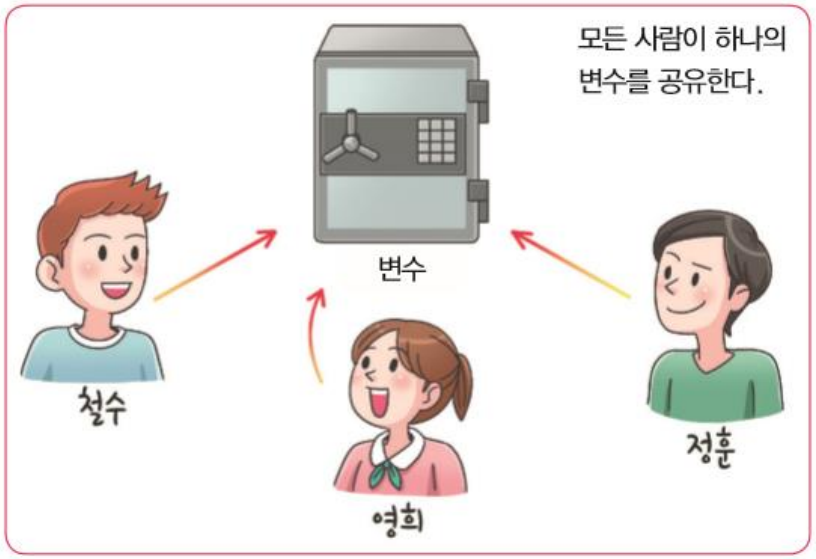
- 객체를 함수로 전달할 때

```
# printAreas()을 호출하여서 객체의 내용이 변경되는지를 확인한다.
myRect = Rectangle();
count = 5
printAreas(myRect, count)
print("사각형의 변=", myRect.side)
print("반복횟수=", count)
```

In [85]:
```
# printAreas()을 호출하여서 객체의 내용이 변경되는지를 확인한다.
myRect = Rectangle();
count = 5
printAreas(myRect, count)
print("사각형의 변=", myRect.side)
print("반복횟수=", count)
```

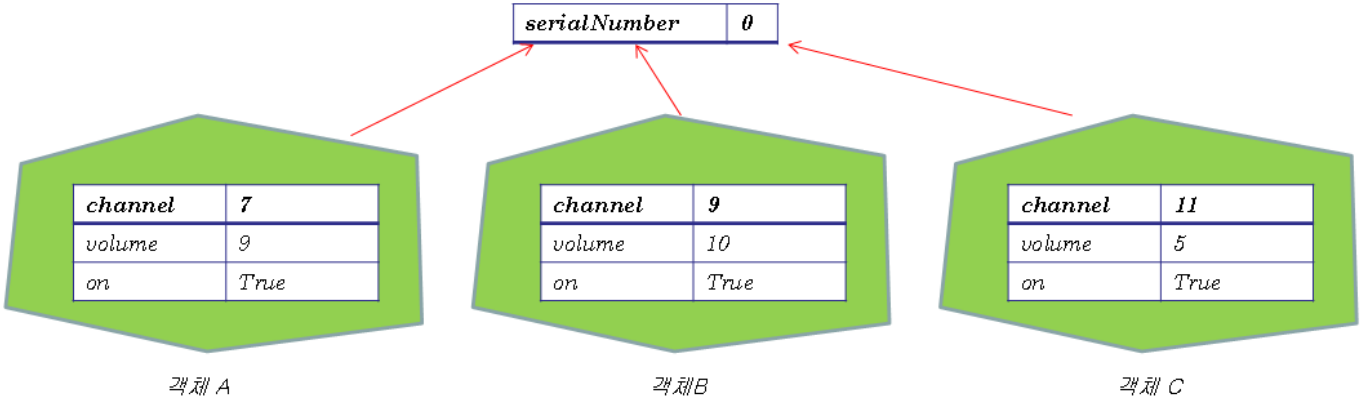
```
0       0
1       1
2       4
3       9
4       16
사각형의 변= 5
반복횟수= 5
```

- 정적 변수
  - 이들 변수는 모든 객체를 통틀어서 하나만 생성되고 모든 객체가 이것을 공유하게 된다. 이러한 변수를 정적 멤버 또는 클래스 멤버(class member)라고 한다.



정적 변수

```
class Television:
        serialNumber = 0          # 이것이 정적 변수이다.
        def __init__(self):
                Television.serialNumber += 1
        self.number = Television.serialNumber
                ...
```



- 특수 메소드
  - 파이썬에는 연산자(+, −, *, /)에 관련된 **특수 메소드(special method)**가 있다.

```
class Circle:
        ...
        def __eq__(self, other):
                return self.radius == other.radius

c1 = Circle(10)
c2 = Circle(10)
if c1 == c2:
        print("원의 반지름은 동일합니다. ")
```

연산자	메소드	설명
x + y	__add__(self, y)	덧셈
x - y	__sub__(self, y)	뺄셈
x * y	__mul__(self, y)	곱셈
x / y	__truediv__(self, y)	실수나눗셈
x // y	__floordiv__(self, y)	정수나눗셈
x % y	__mod__(self, y)	나머지
divmod(x, y)	__divmod__(self, y)	실수나눗셈과 나머지
x ** y	__pow__(self, y)	지수
x << y	__lshift__(self, y)	왼쪽 비트 이동
x >> y	__rshift__(self, y)	오른쪽 비트 이동
x <= y	__le__(self, y)	less than or equal(작거나 같다)
x < y	__lt__(self, y)	less than(작다)
x >= y	__ge__(self, y)	greater than or equal(크거나 같다)
x > y	__gt__(self, y)	greater than(크다)
x == y	__eq__(self, y)	같다
x != y	__neq__(self, y)	같지않다

OOP 예제

```
class Vector2D :
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Vector2D(self.x + other.x, self.y + other.y)

    def __sub__(self, other):
        return Vector2D(self.x - other.x, self.y - other.y)

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __str__(self):
        return '(%g, %g)' % (self.x, self.y)

u = Vector2D(0,1)
v = Vector2D(1,0)
w = Vector2D(1,1)
a = u + v
print( a)
```

OOP 예제

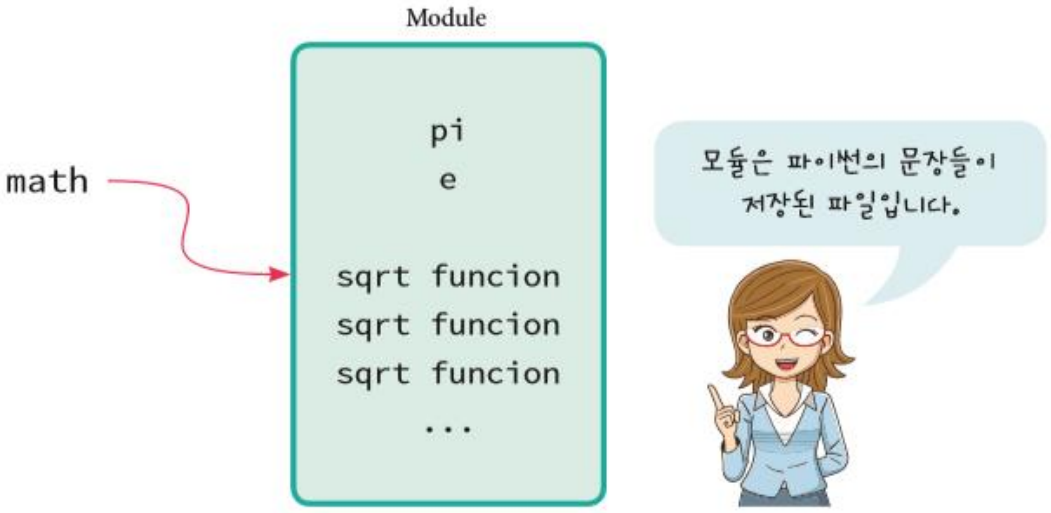
- 파이썬에서의 변수의 종류
  - 지역 변수 – 함수 안에서 선언되는 변수
  - 전역 변수 – 함수 외부에서 선언되는 변수
  - 인스턴스 변수 – 클래스 안에 선언된 변수, 앞에 self.가 붙는다.

OOP 예제

- 핵심 정리
  - 클래스는 속성과 동작으로 이루어진다. 속성은 인스턴스 변수로 표현되고 동작은 메소드로 표현된다.
  - 객체를 생성하려면 생성자 메소드를 호출한다. 생성자 메소드는 __init__() 이름의 메소드이다.
  - 인스턴스 변수를 정의하려면 생성자 메소드 안에서 self.변수이름 과 같이 생성한다.

모듈

- 파이썬에서 **모듈(module)**이란 함수나 변수 또는 클래스들을 모아 놓은 파일이다.



- 모듈 작성하기

fibo.py

```
# 피보나치 수열 모듈

def fib(n):           # 피보나치 수열 출력
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()

def fib2(n):          # 피보나치 수열을 리스트로 반환
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

모듈

- 모듈 사용하기

```
>>> import fibo

>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987

>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

>>> fibo.__name__
'fibo'
```

- 모듈 실행하기

C> python fibo.py <arguments>

```
...
if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

C> python fibo.py 50
1 1 2 3 5 8 13 21 34

모듈

- 모듈 탐색 경로
  1. 입력 스크립트가 있는 디렉토리(파일이 지정되지 않으면 현재 디렉토리)
  2. PYTHONPATH 환경 변수
  3. 설치에 의존하는 디폴트값

모듈

- 유용한 모듈
  - 프로그래밍에서 중요한 하나의 원칙은 이전에 개발된 코드를 적극적으로 재활용하자는 것

- copy 모듈

```
import copy
colors = ["red", "blue", "green"]
clone = copy.deepcopy(colors)

clone[0] = "white"
print(colors)
print(clone)
```

```
['red', 'blue', 'green']
['white', 'blue', 'green']
```

- keyword 모듈

```
import keyword

name = input("변수 이름을 입력하시오: ")

if keyword.iskeyword(name):
    print (name, "은 예약어임.")
    print ("아래는 키워드의 전체 리스트임: ")
    print (keyword.kwlist)
else:
    print (name, "은 사용할 수 있는 변수이름임.")
```

```
변수 이름을 입력하시오: for
for 은 예약어임.
아래는 키워드의 전체 리스트임:
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del',
'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda',
'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

- random 모듈

```
>>> import random
>>> print(random.randint(1, 6))
6
>>> print(random.randint(1, 6))
3

>>> import random
>>> print(random.random()*100)
81.1618515880431

>>> myList = [ "red", "green", "blue" ]
>>> random.choice(myList)
'blue'
```

- os 모듈

```
>>> import os
>>> os.system("calc")
```

```
>>> os.getcwd()
'D:\\'
>>> os.chdir("D:\\tmp")
>>> os.getcwd()
'D:\\tmp'

>>> os.listdir(".")
[ 'chap01', 'chap02', 'chap03', 'chap04', 'chap05', 'chap06', 'chap07',
'chap08', 'chap09', 'chap10', 'chap11', 'chap12', 'chap13', 'chap14', 'chap15',
'chap16', 'chap17', 'chap18', 'chap19', 'chap20' ]
```

- sys 모듈

```
>>> import sys
>>> sys.prefix # 파이썬이 설치된 경로
'C:\\Users\\chun\\AppData\\Local\\Programs\\Python\\Python35-32’

>>> sys.executable
'C:\\Users\\chun\\AppData\\Local\\Programs\\Python\\Python35-
32\\pythonw.exe’
```

- time 모듈

```
>>> import time
>>> time.time()
1461203464.6591916
```

- calendar 모듈

```
import calendar

cal = calendar.month(2016, 8)
print(cal)
```

```
    August 2016
Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

모듈 예제

- 동전 던지기 게임
  - 하나의 예제로 동전 던지기 게임을 파이썬으로 작성해보자. random 모듈을 사용한다.

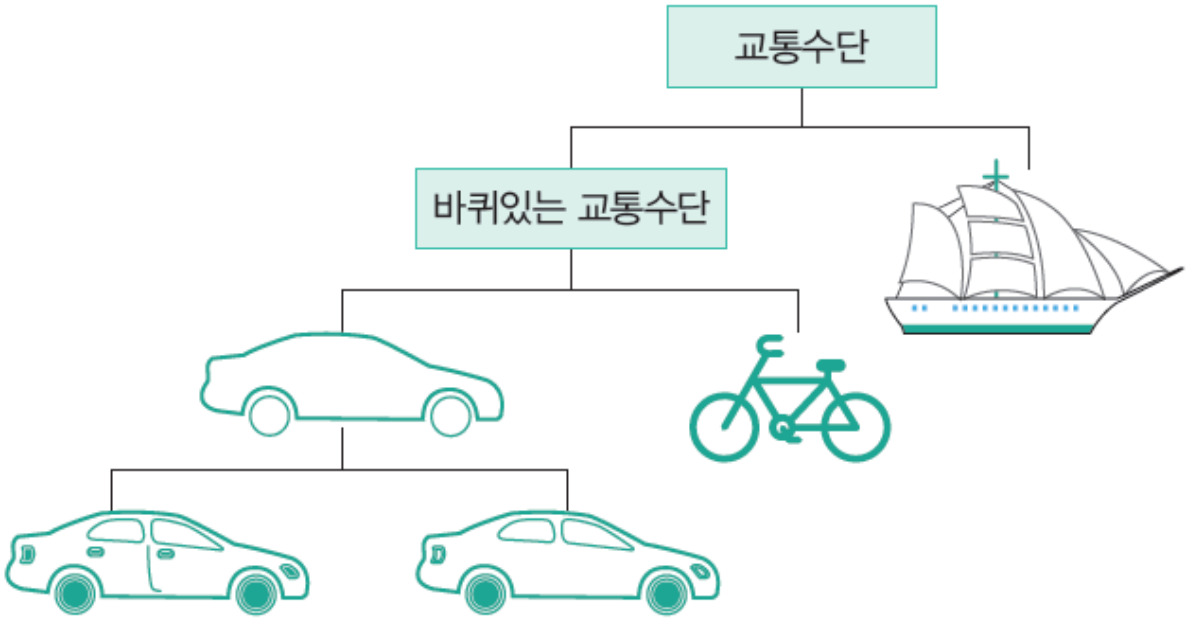
```
동전 던지기를 계속하시겠습니까?( yes, no) yes
head
동전 던지기를 계속하시겠습니까?( yes, no) yes
head
동전 던지기를 계속하시겠습니까?( yes, no) yes
tail
동전 던지기를 계속하시겠습니까?( yes, no) yes
tail
동전 던지기를 계속하시겠습니까?( yes, no) yes
head
동전 던지기를 계속하시겠습니까?( yes, no) no
```

모듈 솔루션

```
import random
myList = [ "head", "tail" ]

while (True):
        response = input("동전 던지기를 계속하시겠습니까?( yes, no) ");
        if response == "yes":
                coin = random.choice(myList)
                print (coin)
        else :
                break
```

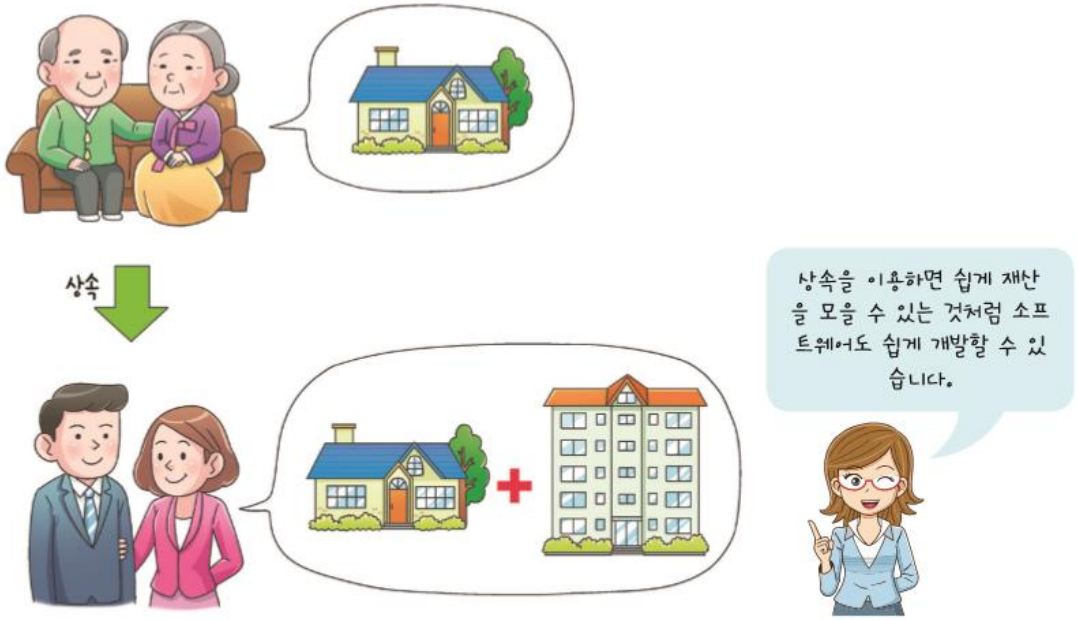
- 핵심정리
  - 파이썬에는 어떤 객체에도 적용이 가능한 내장 함수가 있다. len()나 max()와 같은 함수들을 잘 사용하면 프로그래밍이 쉬워진다.
  - 클래스를 정의할 때 __iter__(self)와 __next__(self) 메소드만 정의하면 이터레이터가 된다. 이터레이터는 for 루프에서 사용할 수 있다.
  - 연산자 오버로딩은 +나 -와 같은 연산자들을 클래스에 맞추어서 다시 정의하는 것이다. 연산자에 해당되는 메소드(예를 들어서 __add__(self, other))를 클래스 안에서 정의하면 된다.

- 상속이란?
  - 상속은 클래스를 정의할 때 부모 클래스를 지정하는 것이다. 자식 클래스는 부모 클래스의 메소드와 변수들을 사용할 수 있다.



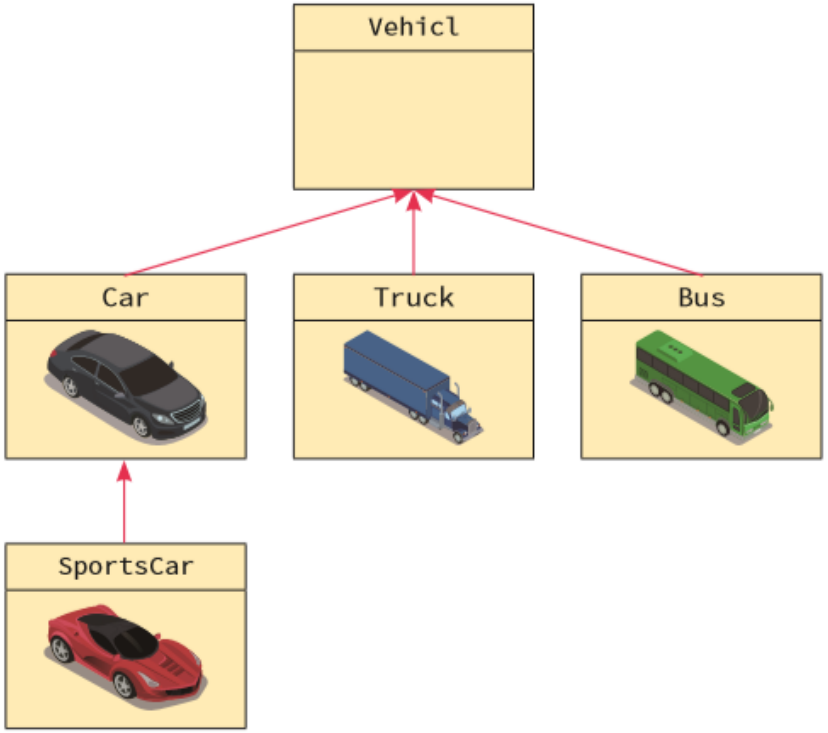
상속과 다형성

- 상속(inheritance)은 기존에 존 재하는 클래스로부터 코드와 데이터를 이어받고 자신이 필요한 기능을 추가하는 기 법이다.



상속과 다형성

- 상속의 예

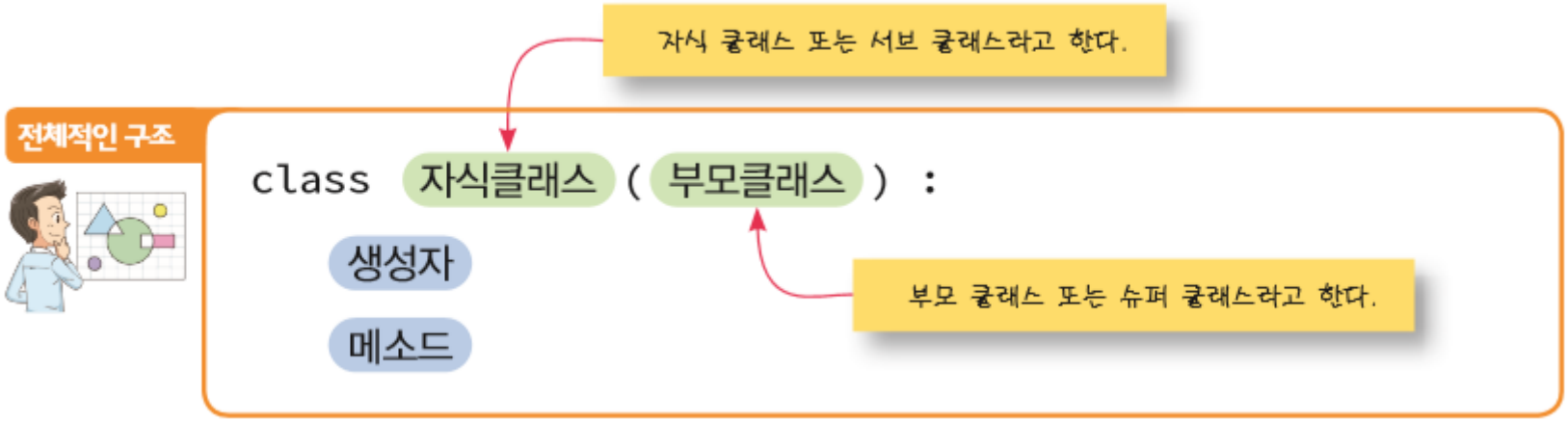


상속과 다형성

- 상속과 is-a 관계
  - 객체 지향 프로그래밍에서는 상속이 클래스 간의 “is-a” 관계를 생성하는데 사용된다.
    - 푸들은 강아지이다.
    - 자동차는 차량이다.
    - 꽃은 식물이다.
    - 사각형은 모양이다.

상속과 다형성

- 상속 구현하기

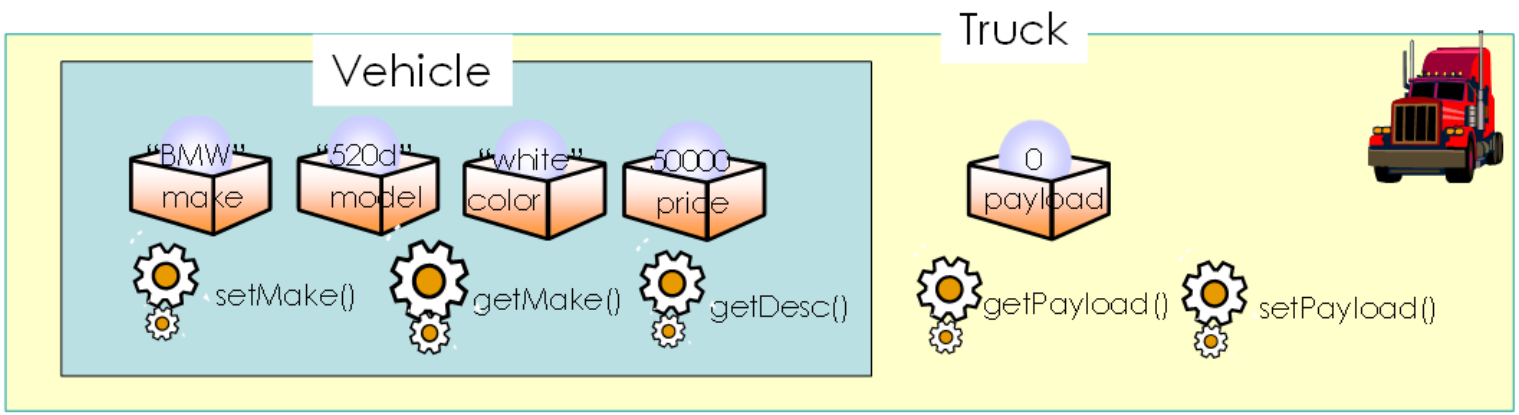


상속과 다형성 예제

```
# 일반적인 운송수단을 나타내는 클래스이다.
class Vehicle:
        def __init__(self, make, model, color,  price):
                self.make = make            # 메이커
                self.model = model          # 모델
                self.color = color          # 자동차의 색상
                self.price = price          # 자동차의 가격

        def setMake(self, make):    # 설정자 메소드
                self.make = make

        def getMake(self):                  # 접근자 메소드
                return self.make

        # 차량에 대한 정보를 문자열로 요약하여서 반환한다.
        def getDesc(self):
                return "차량 =("+str(self.make)+","+\
                        str(self.model)+","+\
                        str(self.color)+","+\
                        str(self.price)+")"
```

상속과 다형성 예제

```
class Truck(Vehicle) :                                  # ①
        def __init__(self, make, model, color, price, payload):
                super().__init__(make, model, color, price)    # ②
                self.payload=payload                           # ③

        def setPayload(self, payload):          # 설정자 메소드
                self.payload=payload

        def getPayload(self):                   # 접근자 메소드
                return self.payload
```



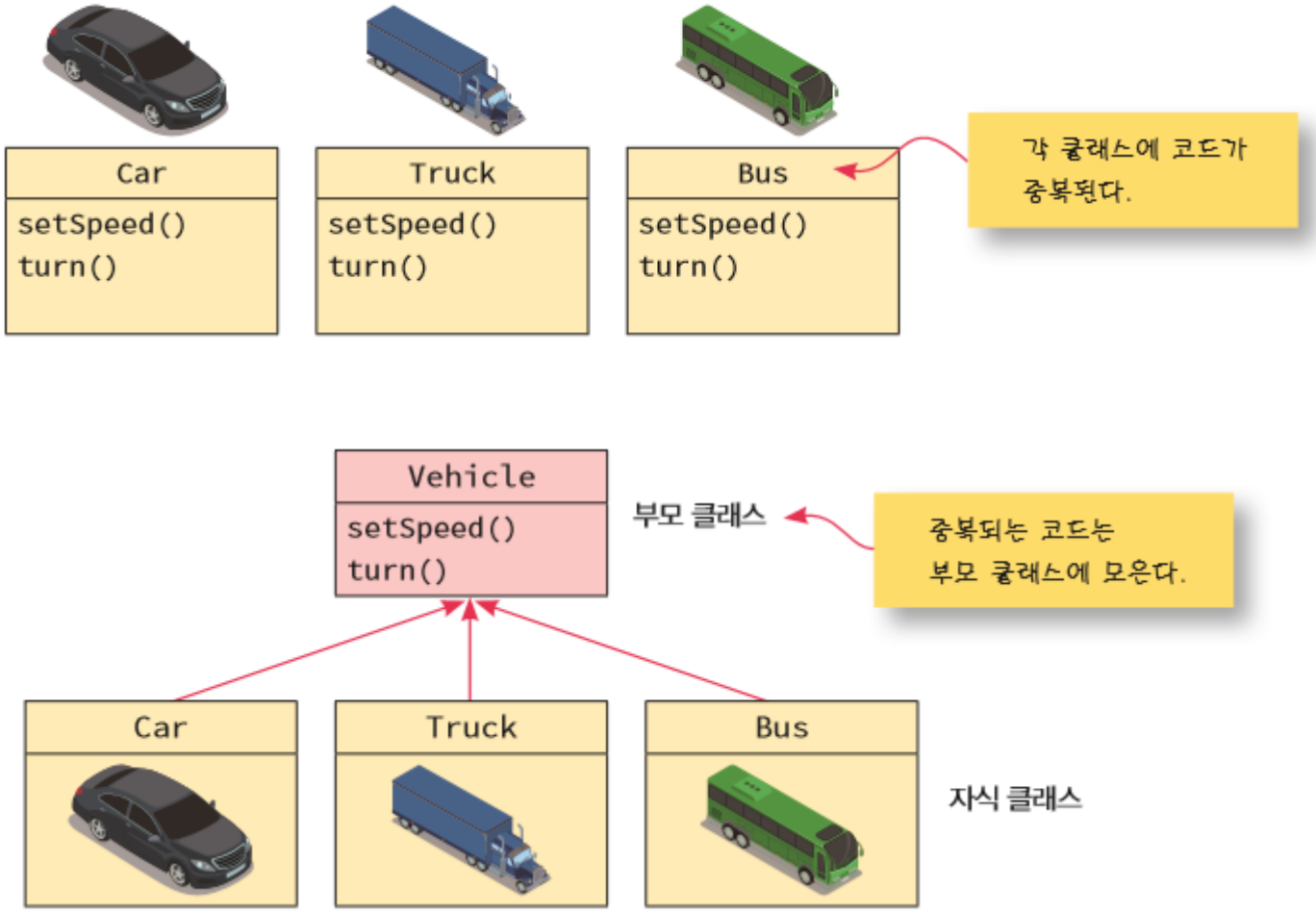
상속과 다형성 예제

```
def main():                                                    # main() 함수 정의
        myTruck = Truck("Tisla", "Model S", "white", 10000, 2000)
        myTruck.setMake("Tesla")                               # 설정자 메소드 호출
        myTruck.setPayload(2000)                               # 설정자 메소드 호출
        print(myTruck.getDesc())                               # 트럭 객체를 문자열로 출력

main()
```

차량 =(Tesla,Model S,white,10000)

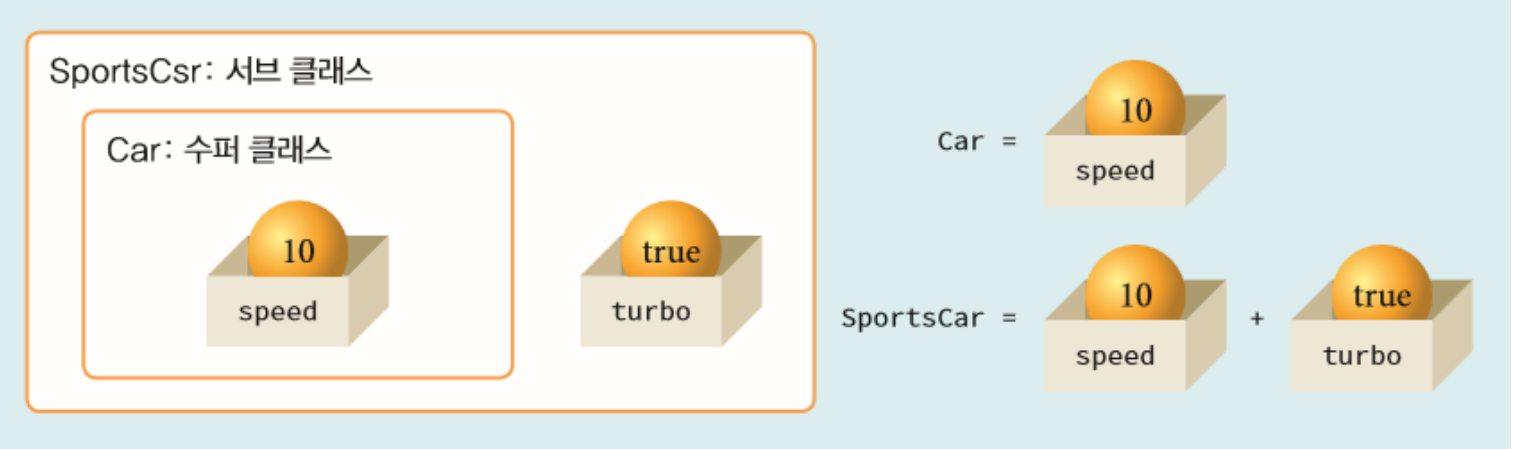
상속과 다형성

- 왜 상속을 사용하는가?



상속과 다형성 실습

- 일반적인 자동차를 나타내는 클래스인 Car 클래스를 상속받아서 수퍼카를 나타내는 클래스인 SportsCar를 작성하는 것이 쉽다. 다음 그림을 참조하여 Car 클래스와 SportsCar 클래스를 작성해보자.



상속과 다형성

```
class Car :
        def __init__(self, speed):
                self.speed = speed
        def setSpeed(self, speed):
                self.speed = speed
        def getDesc(self):
                return "차량 =("+str(self.speed) + ")"

class SportsCar(Car) :
        def __init__(self, speed, turbo):
                super().__init__(speed)
                self.turbo=turbo

        def setTurbo(self, turbo):
                self.turbo=turbo

obj = SportsCar(100, True)
print(obj.getDesc())
obj.setTurbo(False)
```

상속과 다형성 실습

- 상속의 예로 일반적인 다각형을 나타내는 Shape 클래스(x 좌표, y좌표, area(), perimeter())를 작성하고 이것을 상속받아서 사각형을 나타내는 Rectangle 클래스(x 좌표, y좌표, 가로길이, 세로길이, area(), perimeter())를 작성해보자.

```
사각형의 면적 20000
사각형의 둘레 600
```

상속과 다형성

```
class Shape:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def area(self):
        print("계산할 수 없음!")
    def perimeter(self):
        print("계산할 수 없음!")

class Rectangle(Shape):
    def __init__(self, x, y, w, h):
        super().__init__(x, y)
        self.w = w
        self.h = h

    def area(self):
        return self.w*self.h
    def perimeter(self):
        return 2*(self.w+self.h)

r = Rectangle(0, 0, 100, 200)

print("사각형의 면적", r.area())
print("사각형의 둘레", r.perimeter())
```

상속과 다형성 실습

- 일반적인 사람을 나타내는 Person 클래스를 정의한다. Person 클래스를 상속받아서 학생을 나타내는 클래스 Student와 선생님을 나타내는 클래스 Teacher를 정의한다.

```
이름=홍길동
주민번호=12345678
수강과목=['자료구조']
평점=0
이름=김철수
주민번호=123456790
강의과목=['Python']
월급=3000000
```

상속과 다형성

```
class Person:
    def __init__(self, name, number):
        self.name = name
        self.number = number

class Student(Person):
    UNDERGRADUATE=0
    POSTGRADUATE = 1

    def __init__(self, name, number, studentType ):
        super().__init__(name, number)
        self.studentType = studentType
        self.gpa=0
        self.classes = []

    def enrollCourse(self, course):
        self.classes.append(course)

    def __str__(self):
        return  "\n이름="+self.name+ "\n주민번호="+self.number+\
            "\n수강과목="+ str(self.classes)+ "\n평점="+str(self.gpa)
```

상속과 다형성

```
class Teacher(Person):
    def __init__(self, name, number):
        super().__init__(name, number)
        self.courses = []
        self.salary=3000000

    def assignTeaching(self, course):
        self.courses.append(course)

    def __str__(self):
        return  "\n이름="+self.name+ "\n주민번호="+self.number+\
            "\n강의과목="+str(self.courses)+ "\n월급="+str(self.salary)

hong = Student("홍길동", "12345678", Student.UNDERGRADUATE )
hong.enrollCourse("자료구조")
print(hong)

kim  = Teacher("김철수", "123456790")
kim.assignTeaching("Python")
print(kim)
```

상속과 다형성 실습

- 은행계좌
  - 은행 계좌를 나타내는 클래스 BankAccount 클래스를 정의한다.

```
저축예금의 잔액= 10500.0
당좌예금의 잔액= 1890000
```

상속과 다형성

```
class BankAccount:
    def __init__(self, name, number, balance):

        self.balance = balance
        self.name = name
        self.number = number


    def withdraw(self, amount):
        self.balance -= amount
        return self.balance

    def deposit(self, amount):
        self.balance += amount
        return self.balance
```

상속과 다형성

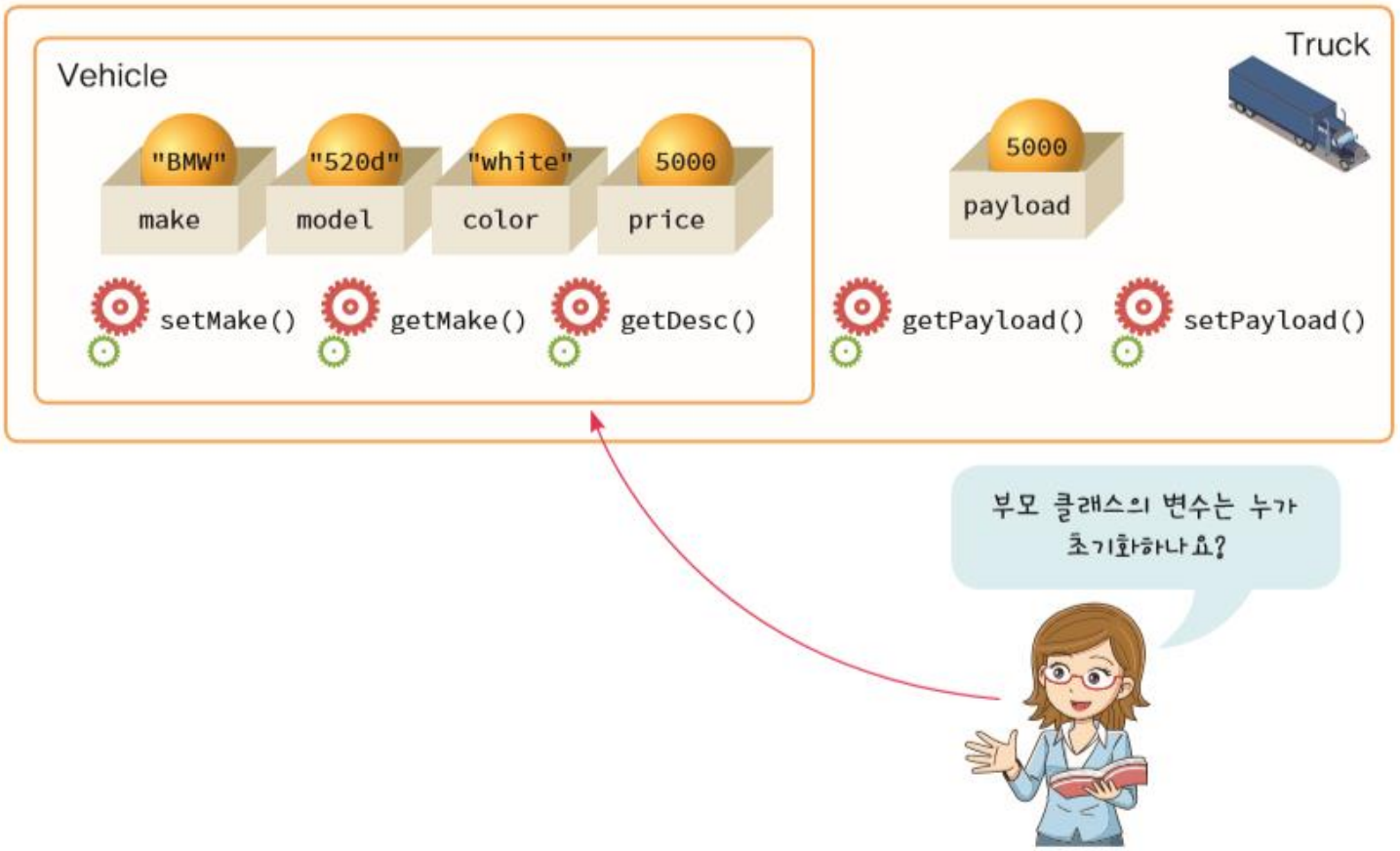
```
class SavingsAccount(BankAccount) :
    def __init__(self,  name, number, balance,  interest_rate):

        super().__init__( name, number, balance)
        self.interest_rate =interest_rate

    def set_interest_rate(self, interest_rate):
        self.interest_rate = interest_rate

    def get_interest_rate(self):
        return self.interest_rate

    def add_interest(self):                    # 예금에 이자를 더한다.
        self.balance += self.balance*self.interest_rate
```

상속과 다형성

```
class CheckingAccount(BankAccount) :
    def __init__(self,  name, number, balance):
        super().__init__( name, number, balance)
        self.withdraw_charge = 10000        # 수표 발행 수수료

    def withdraw(self, amount):
        return BankAccount.withdraw(self, amount + self.withdraw_charge)

a1 = SavingsAccount("홍길동", 123456, 10000, 0.05)
a1.add_interest()
print("저축예금의 잔액=", a1.balance)

a2 = CheckingAccount("김철수", 123457, 2000000)
a2.withdraw(100000)
print("당좌예금의 잔액=", a2.balance)
```

상속과 다형성

- 부모 클래스의 생성자 호출



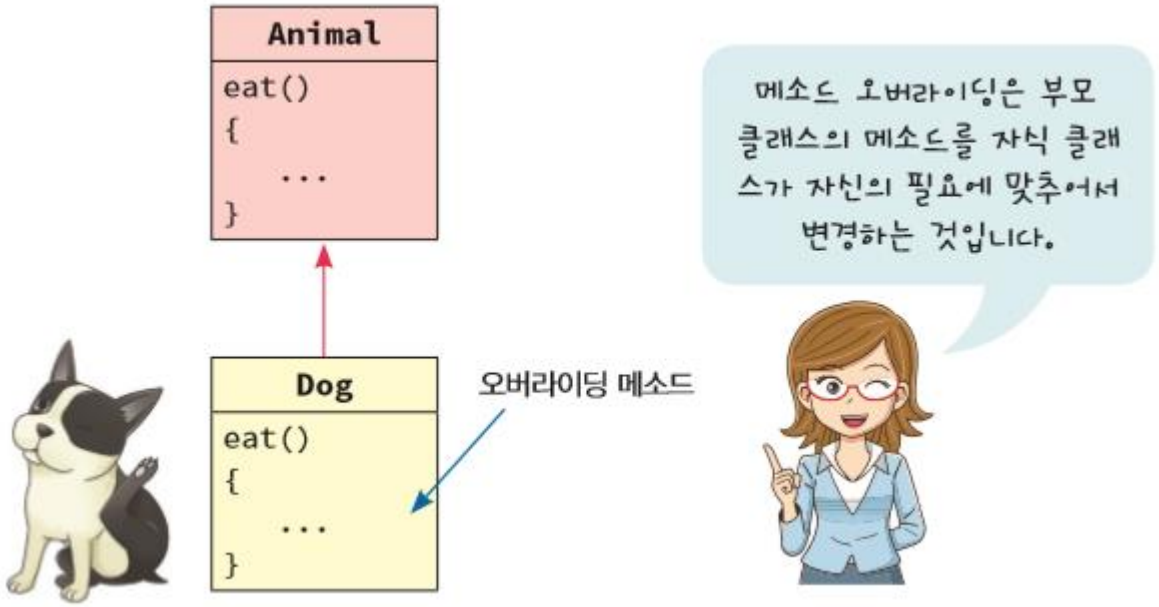
상속과 다형성

- 부모 클래스의 생성자를 명시적으로 호출

```
class ChildClass(ParentClass) :
        def __init__(self):
                super().__init__()
                ...
```

상속과 다형성

- 메소드 오버라이딩
  - "자식 클래스의 메소드가 부모 클래스의 메소드를 오버라이드(재정의)한다"고 말한다.



상속과 다형성 예제

```
class Animal:
    def __init__(self, name=""):
        self.name=name
    def eat(self):
        print("동물이 먹고 있습니다. ")

class Dog(Animal):
    def __init__(self):
            super().__init__()
    def eat(self):
        print("강아지가 먹고 있습니다. ")

d = Dog();
d.eat()
```

강아지가 먹고 있습니다.

상속과 다형성 실습

- 직원과 매니저
  - 회사에 직원(Employee)과 매니저(Manager)가 있다. 직원은 월급만 있지만 매니저는 월급외에 보너스가 있다고 하자. Employee 클래스를 상속받아서 Manager 클래스를 작성한다. Employee 클래스의 getSalary()는 Manager 클래스에서 재정의된다.

```
이름: 김철수; 월급: 2000000; 보너스: 1000000
```

상속과 다형성

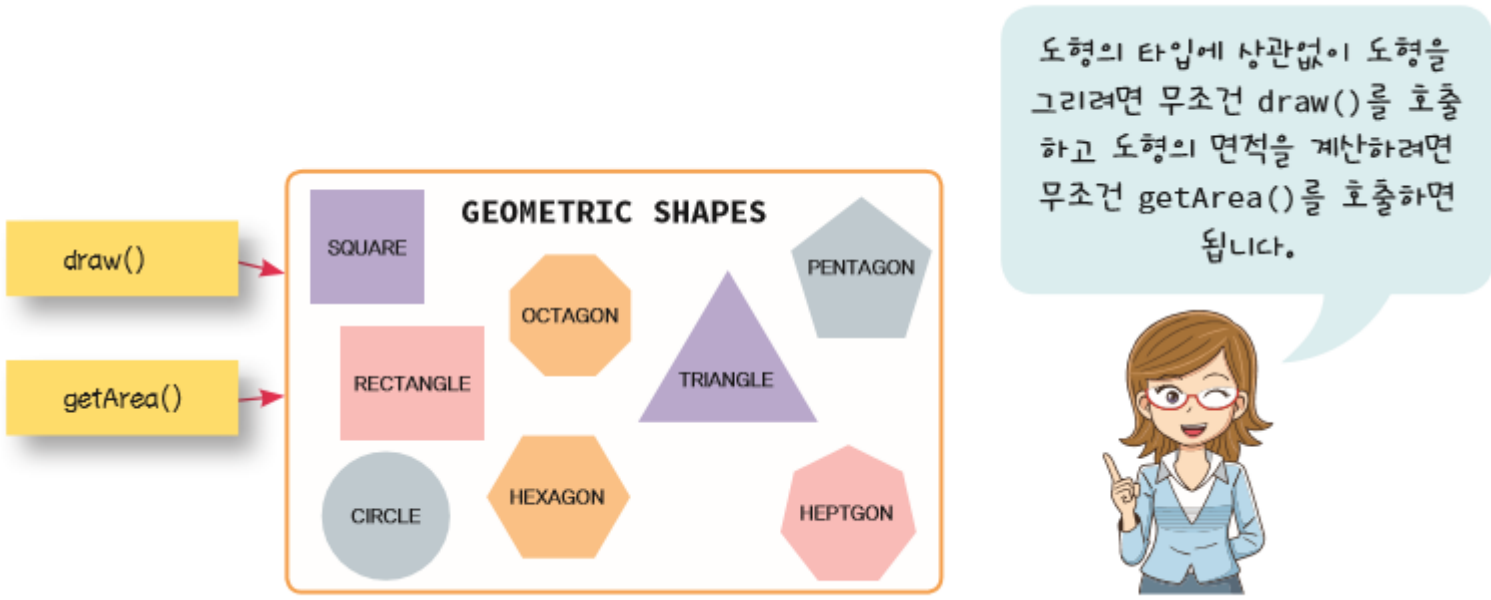
```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def  getSalary(self):
        return salary

class Manager(Employee):
    def __init__(self, name, salary, bonus):
        super().__init__(name, salary)
        self.bonus =bonus

    def  getSalary(self):
        salary = super().getSalary()
        return salary + self.bonus

    def __repr__(self):
        return "이름: "+ self.name+ "; 월급: "+ str(self.salary)+\
               "; 보너스: "+str(self.bonus)

kim = Manager("김철수", 2000000, 1000000)
print(kim)
```

상속과 다형성

- 다형성
  - **다형성(polymorphism)**은 **"많은(poly)+모양(morph)"**이라는 의미로서 주로 프로그래밍 언어에서 하나의 식별자로 다양한 타입(클래스)을 처리하는 것을 의미한다.

상속과 다형성

- 다형성의 예



상속과 다형성

- 상속과 다형성

```
class Animal:
    def __init__(self, name):
        self.name = name
    def speak(self):
        return '알 수 없음'

class Dog(Animal):
    def speak(self):
        return '멍멍!'

class Cat(Animal):
    def speak(self):
        return '야옹!'

animalList = [Dog('dog1'),
              Dog('dog2'),
              Cat('cat1')]

for a in animalList:
    print (a.name + ': ' + a.speak())
```

상속과 다형성

- Lab: Vehicle와 Car, Truck
  - 일반적인 운송수단을 나타내는 Vehicle 클래스를 상속받아서 Car 클래스와 Truck 클래스를 작성해보자.

```
truck1: 트럭을 운전합니다.
truck2: 트럭을 운전합니다.
car1: 승용자를 운전합니다.
```

상속과 다형성

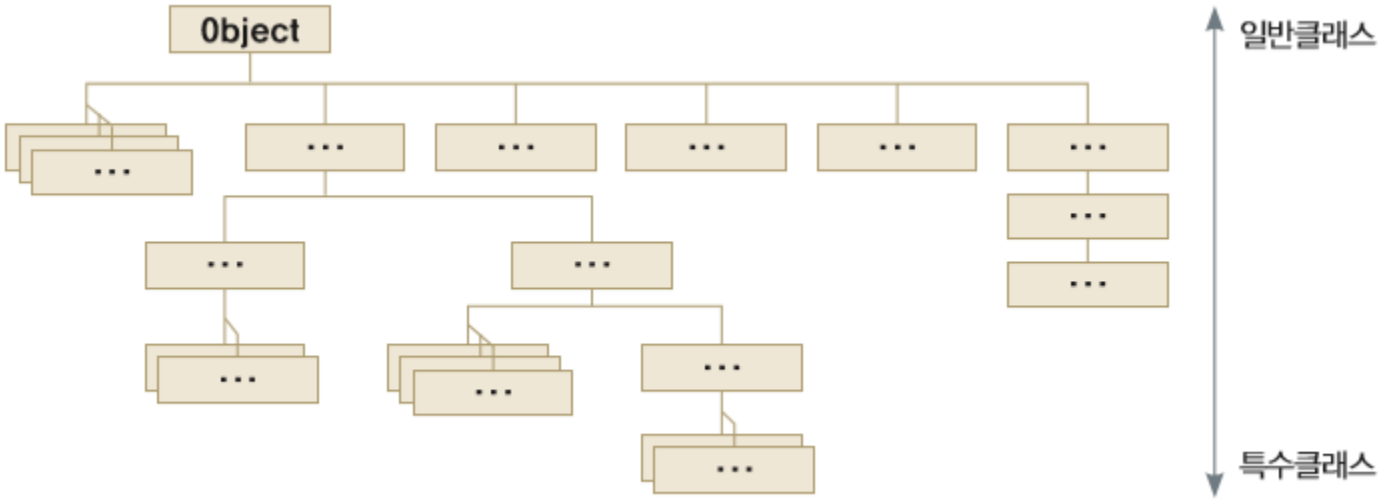
```
class Vehicle:
    def __init__(self, name):
        self.name = name

    def drive(self):
        raise NotImplementedError("이것은 추상메소드입니다. ")

    def stop(self):
        raise NotImplementedError("이것은 추상메소드입니다. ")
```

상속과 다형성

```
class Car(Vehicle):
    def drive(self):
        return '승용자를 운전합니다. '

    def stop(self):
        return '승용자를 정지합니다. '


class Truck(Vehicle):
    def drive(self):
        return '트럭을 운전합니다. '

    def stop(self):
        return '트럭을 정지합니다. '

cars = [Truck('truck1'), Truck('truck2'),  Car('car1')]

for car in cars:
    print( car.name + ': ' + car.drive())
```

상속과 다형성

- Object 클래스
  - 모든 클래스의 맨 위에는 object 클래스가 있다고 생각하면 된다.



상속과 다형성

- Object 클래스의 메소드

메소드
`__init__ ( self [,args...] )` 생성자 (예) `obj = className(args)`
`__del__( self )` 소멸자 (예) `del obj`
`__repr__( self )` 객체 표현 문자열 반환 (예) `repr(obj)`

메소드
`__str__( self )` 문자열 표현 반환 (예) `str(obj)`
`__cmp__ ( self, x )` 객체 비교 (예) `cmp(obj, x)`

상속과 다형성

- 예

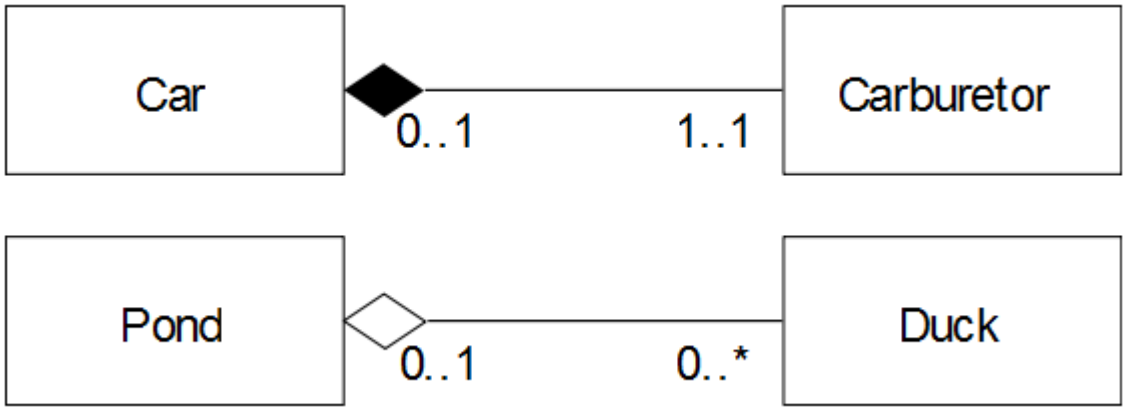
```
class Book:
        def __init__(self, title, isbn):
                self.__title = title
                self.__isbn = isbn
        def __repr__(self):
                return "ISBN: "+ self.__isbn+ "; TITLE: "+ self.__title

book = Book("The Python Tutorial", "0123456")
print(book)
```

ISBN: 0123456; TITLE: The Python Tutorial

상속과 다형성

- 클래스 관계

  is-a 관계: 상속
  has-a 관계



상속과 다형성

- 예

```
class Animal(object):
    pass

class Dog(Animal):
    def __init__(self, name):
        self.name = name

class Person(object):
    def __init__(self, name):
        self.name = name
        self.pet = None


dog1 = Dog("dog1")
person1 = Person("홍길동")
person1.pet = dog1
```

상속과 다형성

- 핵심 정리
  - 상속은 다른 클래스를 재사용하는 탁월한 방법이다. 객체와 객체간의 is-a 관계가 성립된다면 상속을 이용하도록 하자.
  - 상속을 사용하면 중복된 코드를 줄일 수 있다. 공통적인 코드는 부모 클래스를 작성하여 한 곳으로 모으도록 하자.
  - 상속에서는 부모 클래스의 메소드를 자식 클래스가 재정의할 수 있다. 이것을 메소드 오버라이딩이라고 한다.

상속과 다형성

상속과 다형성