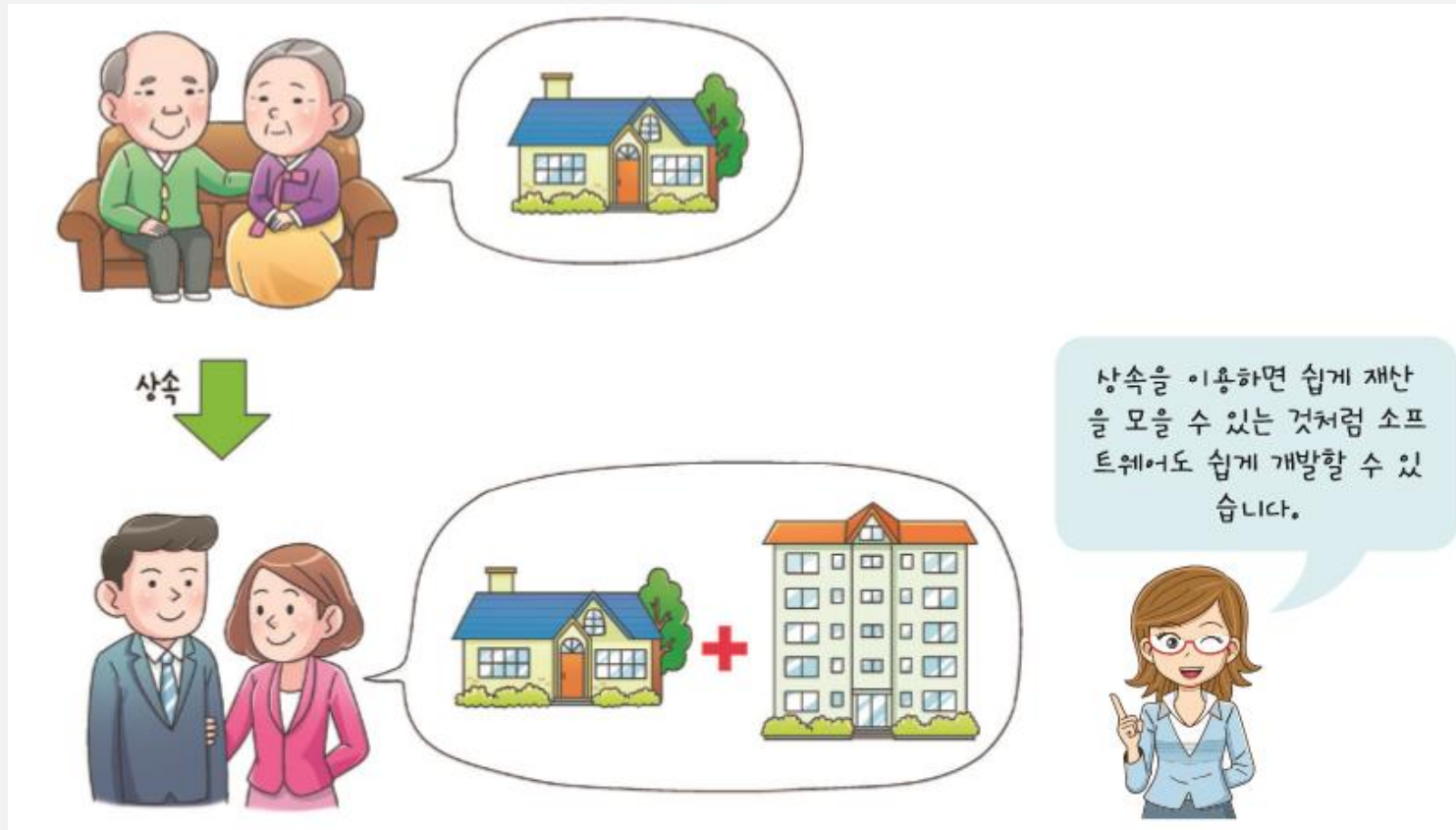


상속(Inheritance)과 다형성(Polymorphism)

By 박수현

상속(inheritance)

- 기존에 존재하는 (부모)class로부터 method와 attribute를 상속받아 사용하거나 자식 class가 필요한 기능을 추가하는 기법



상속 (Inheritance)

- Class를 정의할 때, 부모 class에 있는 속성 및 method를 자식 class가 물려 받아 재사용하는 것
 - Python에서는 method override는 가능하나 **overload**는 허용하지 않음
 - **생성자 overload ?**
 - Override : 상속관계에 있는 자식 class가 부모 class가 정의한 method를 동일한 이름과 동일한 signature로 재정의하여 사용하는 것
 - Overload : 동일한 Class 내에 서로 다른 signature를 가지는 동일한 이름을 갖는 method를 여러 개 정의하여 사용하는 것
- * signature
 - Argument의 개수, type 등
 - Python에서는 일반적인 개념과 다소 상이할 수 있음

상속 구현하기

전체적인 구조



```
class 자식클래스 ( 부모클래스 ) :
```

생성자

메소드

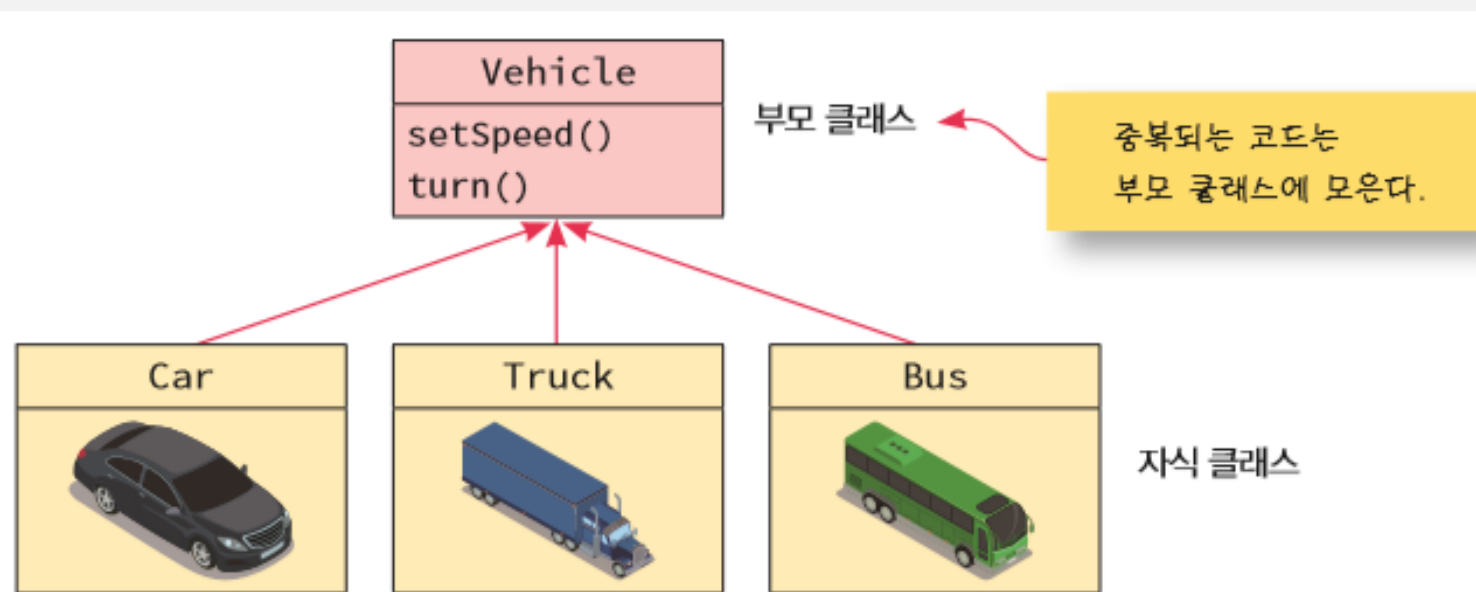
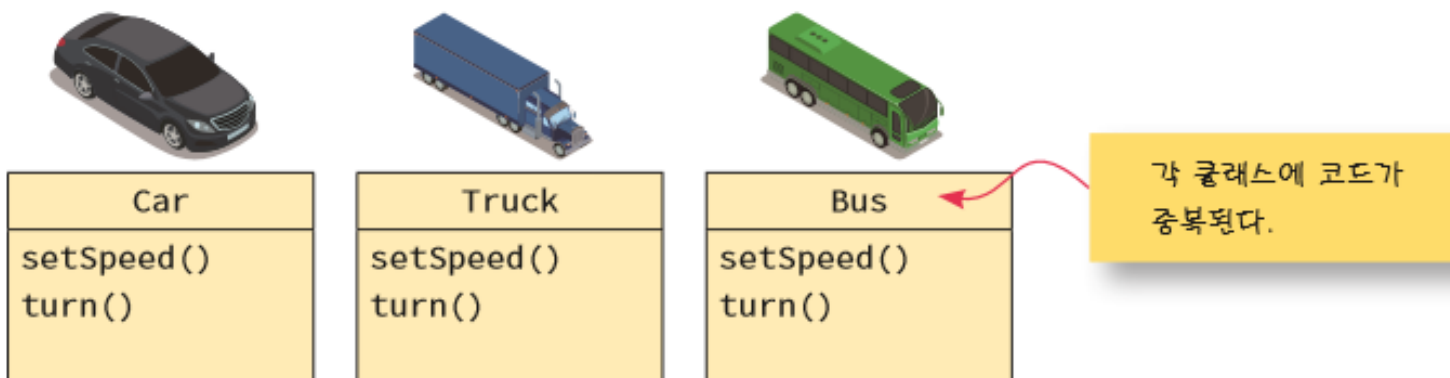
자식 클래스 또는 서브 클래스라고 한다.

부모 클래스 또는 슈퍼 클래스라고 한다.

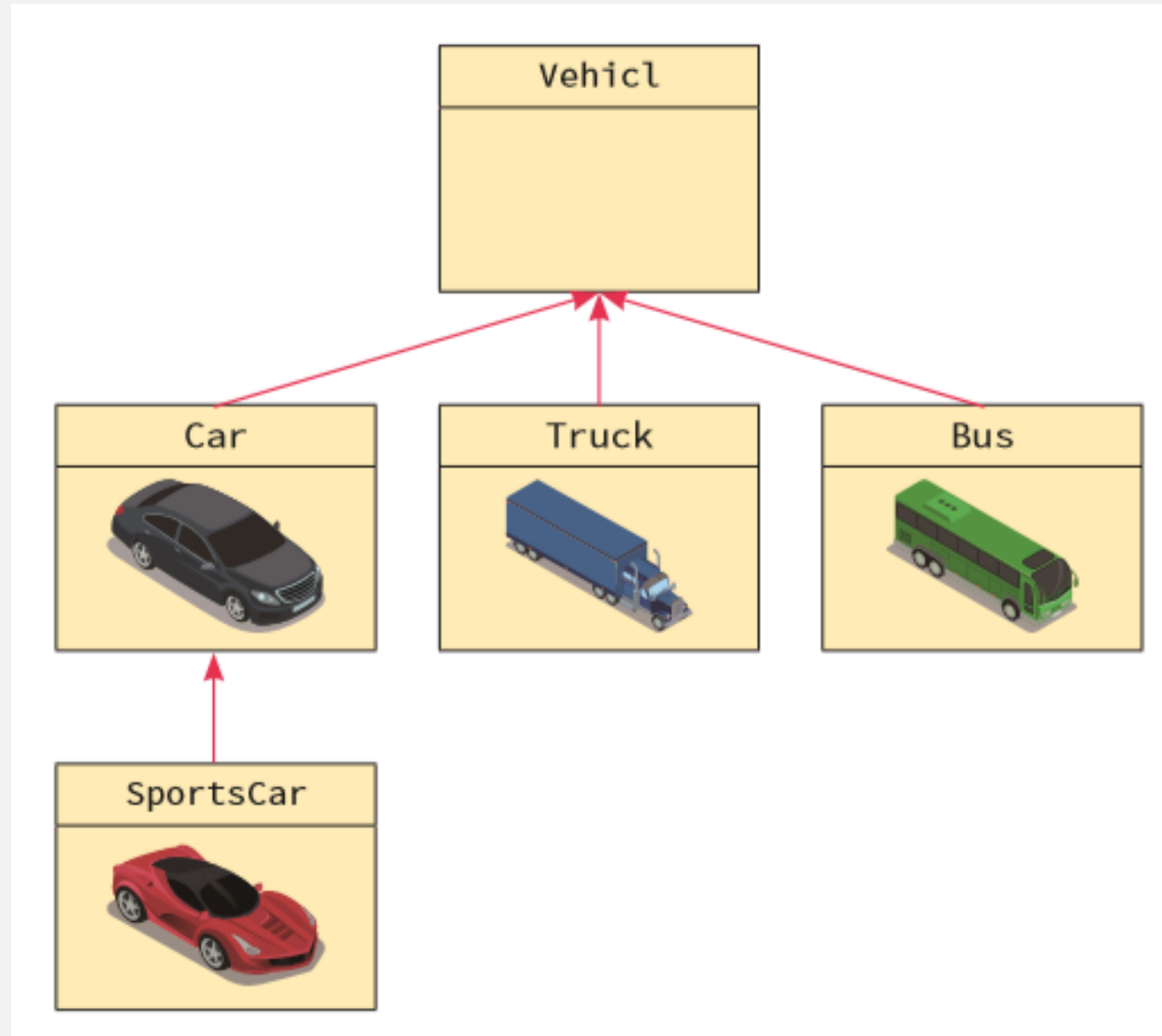
상속과 is-a 관계

- 객체 지향 프로그래밍에서는 상속이 Class 간의 “is-a” 관계를 생성하는데 사용
 - 푸들은 강아지이다.
 - 스포츠카는 차(car)이다.
 - 꽃은 식물이다.
 - 사각형은 모양(shape)이다.

왜 상속을 사용하는가?



상속의 예



예제

일반적인 운송수단을 나타내는 Class이다.

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 getMake(self):            # Getter
        return self.make
```

```
    def setMake(self, make):      # Setter
        self.make = make
```

차량에 대한 정보를 문자열로 요약하여서 반환

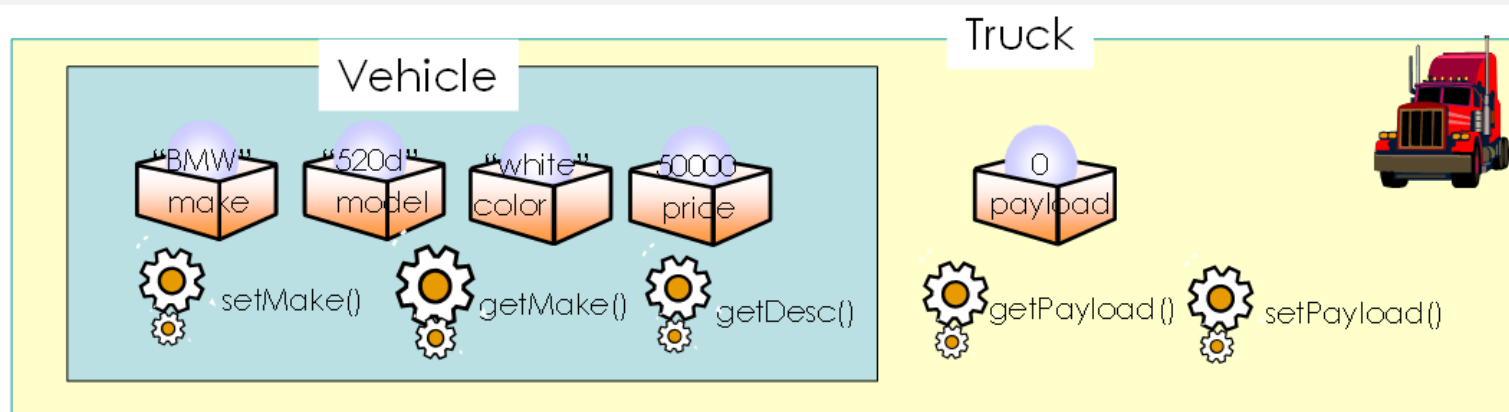
```
    def getDesc(self):
        return "차량=(" + str(self.make) + "," + \
               str(self.model) + "," + \
               str(self.color) + "," + \
               str(self.price) + ")"
```


예제

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

    def setPayload(self, payload):    # Setter
        self.payload=payload

    def getPayload(self):            # Getter
        return self.payload
```



예제

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

main()
```

차량=(Tesla, Model S, white, 10000)

```
class Vehicle:
```

```
    def __init__(self, make, model, color, price):
        print("vi-1) init..")
        self.make = make          # 메이커
        self.model = model        # 모델
        self.color = color        # 자동차의 색상
        self.price = price        # 자동차의 가격
        print("vi-1) self.make = ", self.make)
        print("vi-2) self.model = ", self.model)
        print("vi-3) self.color = ", self.color)
        print("vi-4) self.price = ", self.price)
```

```
    def setMake(self, make):      # setter
        self.make = make
        print("sm-1) self.make = ", self.make)
```

```
    def getMake(self):           # getter
        print("gm-1) getMake()")
        print("gm-2) self.make = ", self.make)
        return self.make
```

```
# 차량에 대한 정보를 문자열로 요약해서 반환
```

```
    def getDesc(self):
        print("gd-1) getDesc()")
        return "차량 = (" + str(self.make) + "," + W
                str(self.model) + "," + W
                str(self.color) + "," + W
                str(self.price) + ")"
```

```
1) main
```

```
m1) main
```

```
ti-1) init..
```

```
vi-1) init..
```

```
vi-1) self.make = Hyundai
```

```
vi-2) self.model = Model S
```

```
vi-3) self.color = white
```

```
vi-4) self.price = 10000
```

```
ti-2) self.payload = 2000
```

```
sm-1) self.make = Tesla
```

```
gm-1) getMake()
```

```
gm-2) self.make = Tesla
```

```
m2) Tesla
```

```
sp-1) setPayload()
```

```
sp-2) self.payload = 2000
```

```
gp-1) getPayload()
```

```
gp-2) self.payload = 2000
```

```
m3) 2000
```

```
gd-1) getDesc()
```

```
m4) 차량 = (Tesla,Model S,white,10000)
```

```

class Truck(Vehicle) :
    def __init__(self, make, model, color, price, payload):
        print("ti-1) init..")
        super().__init__(make, model, color, price)
        self.payload=payload
        print("ti-2) self.payload = ", self.payload)

    def setPayload(self, payload):    # setter
        print("sp-1) setPayload()")
        self.payload=payload
        print("sp-2) self.payload = ", self.payload)

    def getPayload(self):            # getter
        print("gp-1) getPayload()")
        print("gp-2) self.payload = ", self.payload)
        return self.payload

def main():
    print("m1) main")
    myTruck = Truck("Hyundai", "Model S", "white", 10000, 2000)
    myTruck.setMake("Tesla")

    print("m2) ", myTruck.getMake() )

    myTruck.setPayload(2000)
    print("m3) ", myTruck.getPayload())

    print("m4) ", myTruck.getDesc())

#main
print("1) main")
main()

```

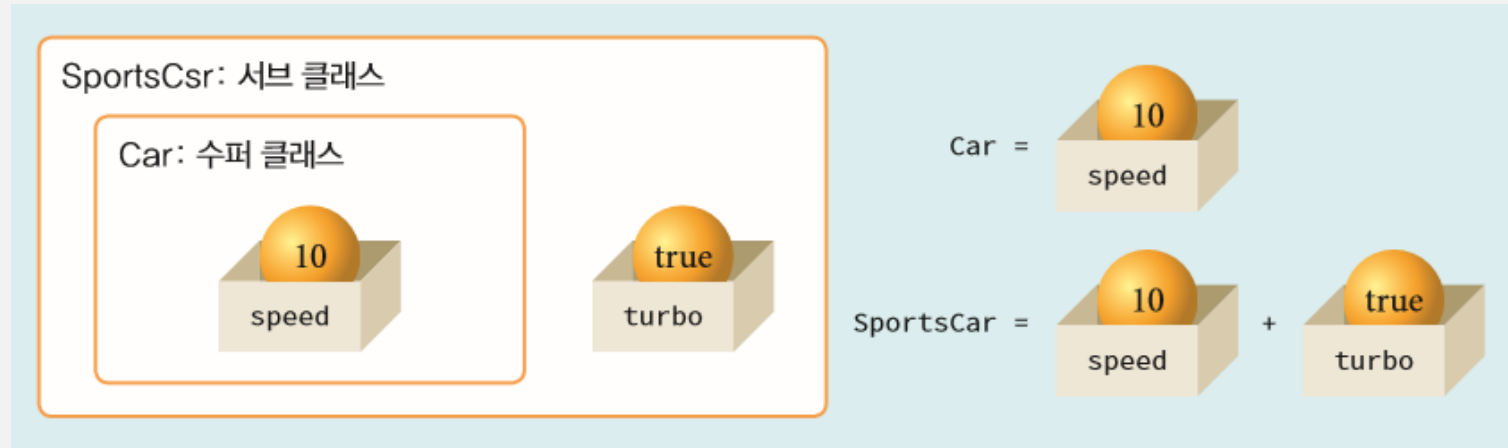
```

1) main
m1) main
ti-1) init..
vi-1) init..
vi-1) self.make = Hyundai
vi-2) self.model = Model S
vi-3) self.color = white
vi-4) self.price = 10000
ti-2) self.payload = 2000
sm-1) self.make = Tesla
gm-1) getMake()
gm-2) self.make = Tesla
m2) Tesla
sp-1) setPayload()
sp-2) self.payload = 2000
gp-1) getPayload()
gp-2) self.payload = 2000
m3) 2000
gd-1) getDesc()
m4) 차량 = (Tesla,Model S,white,10000)

```

Lab: Sportscar Class

- 일반적인 자동차를 나타내는 Class인 Car Class를 상속받아서 자식 class인 SportsCar를 작성. 다음 그림을 참조하여 Car Class와 SportsCar Class를 작성해보자.



Solution

```
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)
```

Lab: 학생과 강사

- 일반적인 사람을 나타내는 Person Class를 정의한다. Person Class를 상속 받아서 학생을 나타내는 Class Student와 선생님을 나타내는 Class Teacher를 정의한다.

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

```
class Person:
```

```
    name = " "
    number = 0
```

```
    def __init__(self, name, number):
        print("ip-1) init in Person() class ")

        self.name = name
        self.number = number

        print("ip-2) self.name =", self.name)
        print("ip-3) self.number =", self.number, "\n")
```

```
class Student(Person):
```

```
    UNDERGRADUATE = 0
    POSTGRADUATE = 1
```

```
    def __init__(self, name, number, s_Type ):
        print("is-1) init in Student() class ")
        super().__init__(name, number)

        self.studentType = s_Type

        self.gpa=0
        self.classes = []

        print("is-2) self.studentType =", self.studentType)
        print("is-3) self.gpa =", self.gpa)
        print("is-4) self.classes =", self.classes, "\n")
```

```
    def enrollCourse(self, course):
        self.classes.append(course)
        print("e-1) self.classes =", self.classes)
```

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

1) main

2) hong 객체생성

is-1) init in Student() class

```
ip-1) init in Person() class
ip-2) self.name = 홍길동
ip-3) self.number = 12345678
```

```
is-2) self.studentType = 0
is-3) self.gpa = 0
is-4) self.classes = []
```

e-1) self.classes = ['자료구조']

```
이름 = 홍길동
주민번호 = 12345678
수강과목 = ['자료구조']
평점 = 0
```

3) kim 객체생성

it-1) init in Teacher() class

```
ip-1) init in Person() class
ip-2) self.name = 김철수
ip-3) self.number = 999999
```

```
it-2) self.courses = []
it-3) self.salary = 3000000
```

a-1) self.courses = ['Python']

```
이름=김철수
주민번호=999999
강의과목=['Python']
월급=3000000
```



```
class Teacher(Person):
```

```
    def __init__(self, name, number):
        print("it-1) init in Teacher() class ")

        super().__init__(name, number)
        self.courses = []
        self.salary = 3000000

        print("it-2) self.courses =", self.courses)
        print("it-3) self.salary =", self.salary)

    def assignTeaching(self, course):
        self.courses.append(course)
        print("Wna-1) self.courses =", self.courses)

    def __str__(self):
        return "Wn이름="+self.name+ "Wn주민번호="+self.number+W
            "Wn강의과목="+str(self.courses)+ "Wn월급="+str(self.salary)
```

```
#main
print("1) main")
print("2) hong 객체생성")
hong = Student("홍길동", "12345678", Student.UNDERGRADUATE)

hong.enrollCourse("자료구조")
print(hong)

print("3) kim 객체생성Wn")
kim = Teacher("김철수", "999999")

kim.assignTeaching("Python")
print(kim)
```

```
1) main
2) hong 객체생성
```

```
is-1) init in Student() class
```

```
ip-1) init in Person() class
ip-2) self.name = 홍길동
ip-3) self.number = 12345678
```

```
is-2) self.studentType = 0
is-3) self.gpa = 0
is-4) self.classes = []
```

```
e-1) self.classes = ['자료구조']
```

```
이름 = 홍길동
주민번호 = 12345678
수강과목 = ['자료구조']
평점 = 0
```

```
3) kim 객체생성
```

```
it-1) init in Teacher() class
```

```
ip-1) init in Person() class
ip-2) self.name = 김철수
ip-3) self.number = 999999
```

```
it-2) self.courses = []
it-3) self.salary = 3000000
```

```
a-1) self.courses = ['Python']
```

```
이름=김철수
주민번호=999999
강의과목=['Python']
월급=3000000
```

Lab: 은행계좌

- 은행 계좌를 나타내는 Class BankAccount Class를 정의한다.

저축예금의 잔액= 10500.0
당좌예금의 잔액= 1890000

Solution

```
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
```

Solution

```
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
```

Solution

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

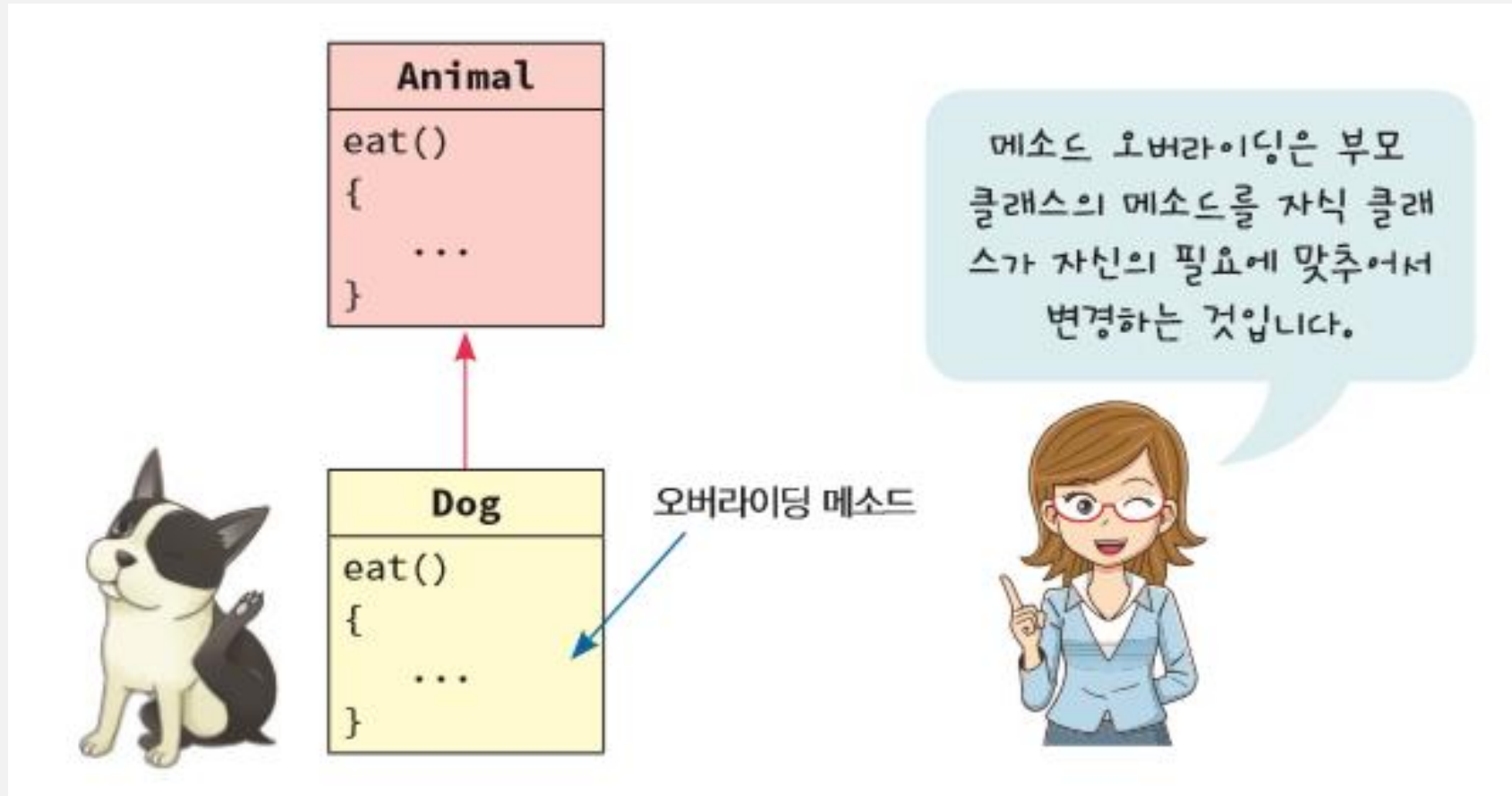
    def withdraw(self, amount):
        return BankAccount.withdraw(self, amount + self.withdraw_charge)

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)
```

Method Overriding

- “자식 Class의 Method가 부모 Class의 Method를 override(재정의)”



Method Overriding 예제 1

```
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):                # method override
        print("강아지가 먹고 있습니다. ")

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

강아지가 먹고 있습니다.

Method Overriding 예제 2

- 상속의 예로 일반적인 다각형을 나타내는 Shape Class(x 좌표, y좌표, area(), perimeter())를 작성하고 이것을 상속받아서 사각형을 나타내는 Rectangle Class(x 좌표, y좌표, 가로길이, 세로길이, area(), perimeter())를 작성
- Override 예제


```
class Shape:
```

```
    def __init__(self, x, y):
```

```
        self.x = x
```

```
        self.y = y
```

```
    def area(self):
```

```
        print("s-a-1) 계산할 수 없음!")
```

```
    def perimeter(self):
```

```
        print("s-p-1) 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): # override
```

```
        print("r-a-1) area() override")
```

```
        return self.w*self.h
```

```
    def perimeter(self): # override
```

```
        print("r-p-1) perimeter() override")
```

```
        return 2*(self.w+self.h)
```

```
#main
```

```
sh = Shape(100, 200)
```

```
print("1) Shape의 면적")
```

```
sh.area()
```

```
print("2) Shape의 둘레")
```

```
sh.perimeter()
```

```
rect = Rectangle(0, 0, 100, 200)
```

```
print("3) 사각형의 면적", rect.area())
```

```
print("4) 사각형의 둘레", rect.perimeter())
```

```
1) Shape의 면적
```

```
s-a-1) 계산할 수 없음!
```

```
2) Shape의 둘레
```

```
s-p-1) 계산할 수 없음!
```

```
r-a-1) area() override
```

```
3) 사각형의 면적 20000
```

```
r-p-1) perimeter() override
```

```
4) 사각형의 둘레 600
```

Method Overriding 예제 3

Lab: 직원과 매니저

- 회사에 직원(Employee)과 매니저(Manager)가 있다. 직원은 월급만 있지만 매니저는 월급외에 보너스가 있다고 하자. Employee Class를 상속받아서 Manager Class를 작성한다. Employee Class의 getSalary()는 Manager Class에서 재정의된다.

이름: 김철수; 월급: 2000000; 보너스: 1000000

Method Overriding 예제 3

```
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)
```

```

class Rectangle():
    def __init__(self, w, h):
        print("ini-1) init")
        print("i-2) w = ", w, ", h = ", h)
        self.w = w
        self.h = h

    def area(self, width):
        print("oa-1) area()")
        print("oa-2) width = ", width)
        area_result = width * width
        return area_result

    def area(self, new_width, new_height):
        print("over load-a-1) area()")
        print("over load-a-2) new_width = ", new_width)
        print("over load-a-3) new_height = ", new_height)
        area_result = new_width * new_height
        return area_result

#main
print("1) main")
rect = Rectangle(100, 200)

print("2) 사각형의 면적", rect.area(20))
print("3) 사각형의 면적", rect.area(20, 20))

```

```

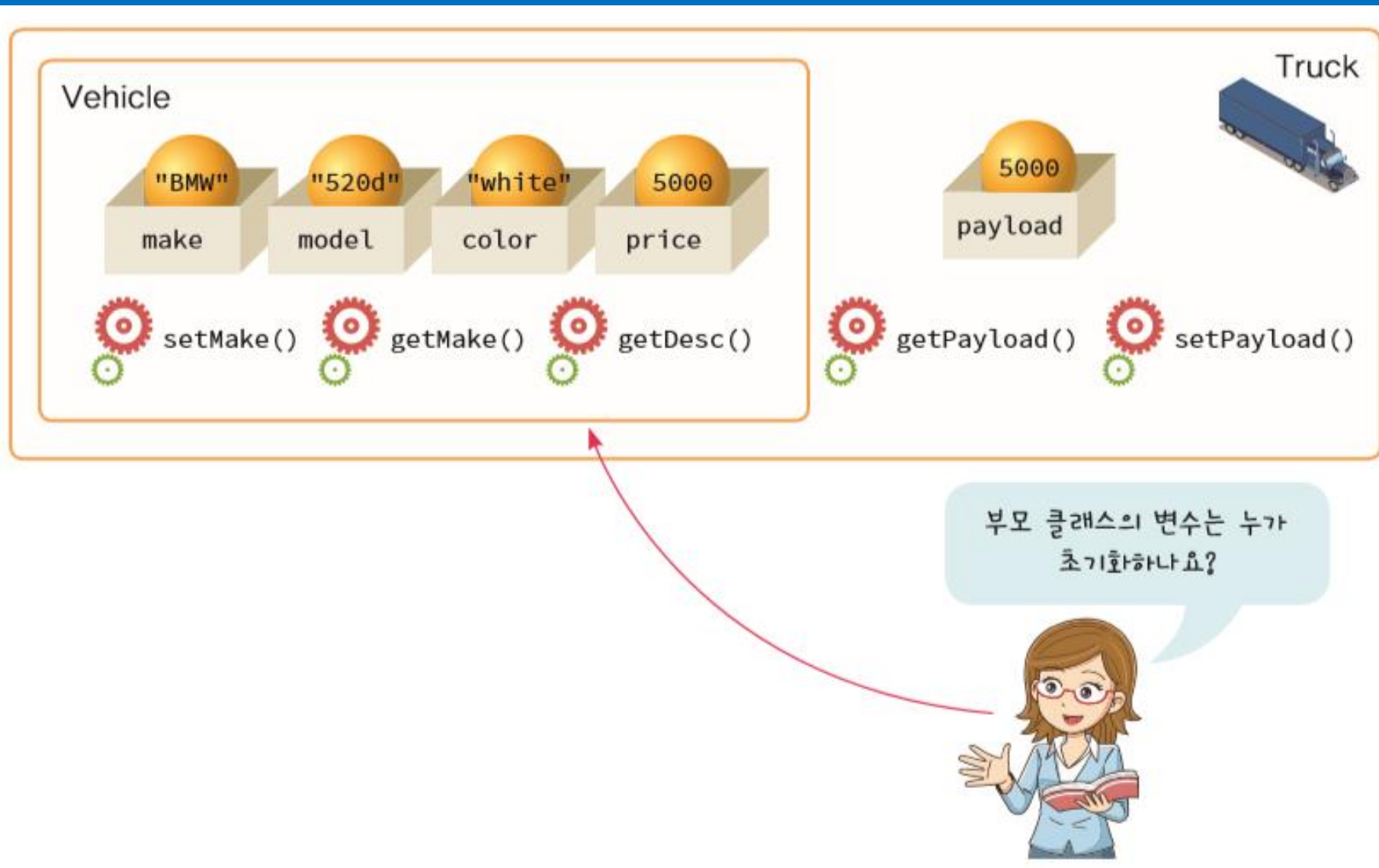
    print("2) 사각형의 면적", rect.area(20))
TypeError: area() missing 1 required positional argument: 'new_height'

```

Method Overload

- 지원하지
않음 !!

부모 Class의 생성자 호출



부모 Class의 생성자를 명시적으로 호출

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

class Animal:

copyright_of_Animal_class = "@Copyright Animal Class."

```
def __init__(self, name, age, weight):
    print("\na-init) Animal Class 생성자")
    print("a-init-1) name = ", name)
    print("a-init-2) age = ", age)
    print("a-init-3) weight = ", weight)
    self._name = name
    self._age = age
    self._w = weight
```

```
def explain_Animal_Class(self):
    print('\na-ex-1) Animal Class에 대한 설명')
    print('a-ex-2) Tiger, Dog, Cat Class의 부모 class입니다.')
```

```
def disp(self):
    pass
```

```
def sound(self):
    print("\na-s-1) Animal Class : 동물의 울음소리를 출력합니다")
```

0) main에서 3개 객체 생성
t-init) Tiger Class 생성자
t-init-1) name = 호돌이
t-init-2) age = 8
t-init-3) weight = 180

a-init) Animal Class 생성자
a-init-1) name = 호돌이
a-init-2) age = 8
a-init-3) weight = 180

d-init) Dog Class 생성자
d-init-1) name = 멍멍이
d-init-2) age = 4
d-init-3) weight = 8

a-init) Animal Class 생성자
a-init-1) name = 멍멍이
a-init-2) age = 4
a-init-3) weight = 8

c-init) Cat Class 생성자
c-init-1) 이름 : 야옹이
c-init-2) 나이 : 3
c-init-3) 몸두게 : 2

a-init) Animal Class 생성자
a-init-1) name = 야옹이
a-init-2) age = 3
a-init-3) weight = 2

1) Tiger 클래스 객체 _tiger_obj 호출
t-d-1) 이름 : 호돌이
t-d-2) 종류 : 호랑이
t-d-3) 나이 : 8
t-d-4) 몸두게 : 180kg
t-s-1) 어흥~~

2) Dog 클래스 객체 _dog_obj 호출
d-d-1) 이름 : 멍멍이
d-d-2) 종류 : 강아지
d-d-3) 나이 : 4
d-d-4) 몸두게 : 8 kg
d-s-1) 멍멍~~

3) Cat 클래스 객체 _cat_obj 호출
c-d-1) 이름 : 야옹이
c-d-2) 종류 : 고양이
c-d-3) 나이 : 3
c-d-4) 몸두게 : 2 kg

c-s-1) 야옹~~

a-ex-1) Animal Class에 대한 설명
a-ex-2) Tiger, Dog, Cat Class의 부모 class입니다.

4) a_cat_obj.copyright_of_Animal_class = @Copyright Animal Class.
5) Baby_Cat 클래스 객체, b_cat 호출
b-init) Baby_Cat Class 생성자
b-init-1) 이름 : 나비아
b-init-2) 나이 : 0.2
b-init-3) 몸두게 : 1
c-init) Cat Class 생성자
c-init-1) 이름 : 나비아
c-init-2) 나이 : 0.2
c-init-3) 몸두게 : 1
a-init) Animal Class 생성자
a-init-1) name = 나비아
a-init-2) age = 0.2
a-init-3) weight = 1
b-d-1) 이름 : 나비아
b-d-2) 종류 : 고양이
b-d-3) 나이 : 0.2
b-d-4) 이름 : 1 kg
b-s-1) 미야요 ~~ 야옹~~
a-ex-1) Animal Class에 대한 설명
a-ex-2) Tiger, Dog, Cat Class의 부모 class입니다.
6) animal_3.copyright_of_Animal_class = @Copyright Animal Class.

```
class Tiger(Animal): # Animal Class 상속
```

```
_kind_of_animal = '호랑이'
```

```
def __init__(self, name, age, weight):  
    print("t-init) Tiger Class 생성자")  
    print("t-init-1) name = ", name)  
    print("t-init-2) age = ", age)  
    print("t-init-3) weight = ", weight)  
    super().__init__(name, age, weight)
```

```
def disp(self): # Animal Class의 disp() method override  
    print('t-d-1) 이름 : '+ self._name)  
    print('t-d-2) 종류 : '+ self._kind_of_animal)  
    print('t-d-3) 나이 : '+ str(self._age))  
    print('t-d-4) 몸무게 : '+ str(self._w)+'kg')  
    self.sound()
```

```
def sound(self): # Animal Class의 sound() method override  
    print('t-s-1) 어흥~~')
```

```
0) main에서 3개 객체 생성  
  
t-init) Tiger Class 생성자  
t-init-1) name = 호돌이  
t-init-2) age = 8  
t-init-3) weight = 180
```

```
a-init) Animal Class 생성자  
a-init-1) name = 호돌이  
a-init-2) age = 8  
a-init-3) weight = 180
```

```
d-init) Dog Class 생성자  
d-init-1) name = 멍멍이  
d-init-2) age = 4  
d-init-3) weight = 8
```

```
a-init) Animal Class 생성자  
a-init-1) name = 멍멍이  
a-init-2) age = 4  
a-init-3) weight = 8
```

```
c-init) Cat Class 생성자  
c-init-1) 이름 : 야옹이  
c-init-2) 나이 : 3  
c-init-3) 몸무게 : 2
```

```
a-init) Animal Class 생성자  
a-init-1) name = 야옹이  
a-init-2) age = 3  
a-init-3) weight = 2
```

```
1) Tiger 클래스 객체 _tiger_obj 호출  
t-d-1) 이름 : 호돌이  
t-d-2) 종류 : 호랑이  
t-d-3) 나이 : 8  
t-d-4) 몸무게 : 180kg  
t-s-1) 어흥~~
```

```
2) Dog 클래스 객체 _dog_obj 호출  
d-d-1) 이름 : 멍멍이  
d-d-2) 종류 : 강아지  
d-d-3) 나이 : 4  
d-d-4) 몸무게 : 8 kg  
d-s-1) 멍멍~~
```

```
3) Cat 클래스 객체 _cat_obj 호출  
  
c-d-1) 이름 : 야옹이  
c-d-2) 종류 : 고양이  
c-d-3) 나이 : 3  
c-d-4) 몸무게 : 2 kg
```

```
c-s-1) 야옹~~
```

```
a-ex-1) Animal Class에 대한 설명  
a-ex-2) Tiger, Dog, Cat Class의 부모 class입니다.
```

```
4) a_cat_obj.copyright_of_Animal_class = @Copyright Animal Class.  
  
5) Baby_Cat 클래스 객체, b_cat 호출  
  
b-init) Baby_Cat Class 생성자  
b-init-1) 이름 : 나비야  
b-init-2) 나이 : 0.2  
b-init-3) 몸무게 : 1  
  
c-init) Cat Class 생성자  
c-init-1) 이름 : 나비야  
c-init-2) 나이 : 0.2  
c-init-3) 몸무게 : 1  
  
a-init) Animal Class 생성자  
a-init-1) name = 나비야  
a-init-2) age = 0.2  
a-init-3) weight = 1  
b-d-1) 이름 : 나비야  
b-d-2) 종류 : 고양이  
b-d-3) 나이 : 0.2  
b-d-4) 이름 : 1 kg  
b-s-1) 미야요 ~~ 야옹~~  
  
a-ex-1) Animal Class에 대한 설명  
a-ex-2) Tiger, Dog, Cat Class의 부모 class입니다.  
  
6) animal_3.copyright_of_Animal_class = @Copyright Animal Class.
```



```
class Dog(Animal): # Animal Class 상속
```

```
_kind_of_animal = '강아지'
```

```
def __init__(self, name, age, weight):
```

```
    print("Wnd-init) Dog Class 생성자")
```

```
    print("d-init-1) name = ", name)
```

```
    print("d-init-2) age = ", age)
```

```
    print("d-init-3) weight = ", weight)
```

```
    super().__init__(name, age, weight)
```

```
def disp(self): # Animal Class의 disp() method override
```

```
    print('d-d-1) 이름 : ' + self._name)
```

```
    print('d-d-2) 종류 : ' + self._kind_of_animal)
```

```
    print('d-d-3) 나이 : ' + str(self._age))
```

```
    print('d-d-4) 몸두게 : ' + str(self._w)+' kg')
```

```
    self.sound()
```

```
def sound(self): # Animal Class의 sound() method override
```

```
    print('d-s-1) 멍멍~~')
```

```
0) main에서 3개 객체 생성
```

```
t-init) Tiger Class 생성자
```

```
t-init-1) name = 호돌이
```

```
t-init-2) age = 8
```

```
t-init-3) weight = 180
```

```
a-init) Animal Class 생성자
```

```
a-init-1) name = 호돌이
```

```
a-init-2) age = 8
```

```
a-init-3) weight = 180
```

```
d-init) Dog Class 생성자
```

```
d-init-1) name = 멍멍이
```

```
d-init-2) age = 4
```

```
d-init-3) weight = 8
```

```
a-init) Animal Class 생성자
```

```
a-init-1) name = 멍멍이
```

```
a-init-2) age = 4
```

```
a-init-3) weight = 8
```

```
c-init) Cat Class 생성자
```

```
c-init-1) 이름 : 야옹이
```

```
c-init-2) 나이 : 3
```

```
c-init-3) 몸두게 : 2
```

```
a-init) Animal Class 생성자
```

```
a-init-1) name = 야옹이
```

```
a-init-2) age = 3
```

```
a-init-3) weight = 2
```

```
1) Tiger 클래스 객체 _tiger_obj 호출
```

```
t-d-1) 이름 : 호돌이
```

```
t-d-2) 종류 : 호랑이
```

```
t-d-3) 나이 : 8
```

```
t-d-4) 몸두게 : 180kg
```

```
t-s-1) 어흥~~
```

```
2) Dog 클래스 객체 _dog_obj 호출
```

```
d-d-1) 이름 : 멍멍이
```

```
d-d-2) 종류 : 강아지
```

```
d-d-3) 나이 : 4
```

```
d-d-4) 몸두게 : 8 kg
```

```
d-s-1) 멍멍~~
```

```
3) Cat 클래스 객체 _cat_obj 호출
```

```
c-d-1) 이름 : 야옹이
```

```
c-d-2) 종류 : 고양이
```

```
c-d-3) 나이 : 3
```

```
c-d-4) 몸두게 : 2 kg
```

```
c-s-1) 야옹~~
```

```
a-ex-1) Animal Class에 대한 설명
```

```
a-ex-2) Tiger, Dog, Cat Class의 부모 class입니다.
```

```
4) a_cat_obj.copyright_of_Animal_class = @Copyright Animal Class.
```

```
5) Baby_Cat 클래스 객체, b_cat 호출
```

```
b-init) Baby_Cat Class 생성자
```

```
b-init-1) 이름 : 나비야
```

```
b-init-2) 나이 : 0.2
```

```
b-init-3) 몸두게 : 1
```

```
c-init) Cat Class 생성자
```

```
c-init-1) 이름 : 나비야
```

```
c-init-2) 나이 : 0.2
```

```
c-init-3) 몸두게 : 1
```

```
a-init) Animal Class 생성자
```

```
a-init-1) name = 나비야
```

```
a-init-2) age = 0.2
```

```
a-init-3) weight = 1
```

```
b-d-1) 이름 : 나비야
```

```
b-d-2) 종류 : 고양이
```

```
b-d-3) 나이 : 0.2
```

```
b-d-4) 이름 : 1 kg
```

```
b-s-1) 미야요 ~~ 야옹~~
```

```
a-ex-1) Animal Class에 대한 설명
```

```
a-ex-2) Tiger, Dog, Cat Class의 부모 class입니다.
```

```
6) animal_3.copyright_of_Animal_class = @Copyright Animal Class.
```

class Cat(Animal): # Animal Class 상속

_kind_of_animal = '고양이'

def __init__(self,name,age,w):

print("#nc-init) Cat Class 생성자")

print('c-init-1) 이름 : ', name)

print('c-init-2) 나이 : ', age)

print('c-init-3) 몸두게 : ', w)

super().__init__(name,age,w)

def disp(self): # Animal Class의 disp() method override

print('Wnc-d-1) 이름 : ' + self._name)

print('c-d-2) 종류 : ' + self._kind_of_animal)

print('c-d-3) 나이 : ' + str(self._age))

print('c-d-4) 몸두게 : ' + str(self._w)+' kg')

self.sound()

def sound(self): # Animal Class의 sound() method override

print('Wnc-s-1) 야옹~~')

0) main에서 3개 객체 생성

t-init) Tiger Class 생성자

t-init-1) name = 호돌이

t-init-2) age = 8

t-init-3) weight = 180

a-init) Animal Class 생성자

a-init-1) name = 호돌이

a-init-2) age = 8

a-init-3) weight = 180

d-init) Dog Class 생성자

d-init-1) name = 멍멍이

d-init-2) age = 4

d-init-3) weight = 8

a-init) Animal Class 생성자

a-init-1) name = 멍멍이

a-init-2) age = 4

a-init-3) weight = 8

c-init) Cat Class 생성자

c-init-1) 이름 : 야옹이

c-init-2) 나이 : 3

c-init-3) 몸두게 : 2

a-init) Animal Class 생성자

a-init-1) name = 야옹이

a-init-2) age = 3

a-init-3) weight = 2

1) Tiger 클래스 객체 _tiger_obj 호출

t-d-1) 이름 : 호돌이

t-d-2) 종류 : 호랑이

t-d-3) 나이 : 8

t-d-4) 몸두게 : 180kg

t-s-1) 어흥~~

2) Dog 클래스 객체 _dog_obj 호출

d-d-1) 이름 : 멍멍이

d-d-2) 종류 : 강아지

d-d-3) 나이 : 4

d-d-4) 몸두게 : 8 kg

d-s-1) 멍멍~~

3) Cat 클래스 객체 _cat_obj 호출

c-d-1) 이름 : 야옹이

c-d-2) 종류 : 고양이

c-d-3) 나이 : 3

c-d-4) 몸두게 : 2 kg

c-s-1) 야옹~~

a-ex-1) Animal Class에 대한 설명

a-ex-2) Tiger, Dog, Cat Class의 부모 class입니다.

4) a_cat_obj.copyright_of_Animal_class = @Copyright Animal Class.

5) Baby_Cat 클래스 객체, b_cat 호출

b-init) Baby_Cat Class 생성자

b-init-1) 이름 : 나비아

b-init-2) 나이 : 0.2

b-init-3) 몸두게 : 1

c-init) Cat Class 생성자

c-init-1) 이름 : 나비아

c-init-2) 나이 : 0.2

c-init-3) 몸두게 : 1

a-init) Animal Class 생성자

a-init-1) name = 나비아

a-init-2) age = 0.2

a-init-3) weight = 1

b-d-1) 이름 : 나비아

b-d-2) 종류 : 고양이

b-d-3) 나이 : 0.2

b-d-4) 이름 : 1 kg

b-s-1) 미야요 ~~ 야옹~~

a-ex-1) Animal Class에 대한 설명

a-ex-2) Tiger, Dog, Cat Class의 부모 class입니다.

6) animal_3.copyright_of_Animal_class = @Copyright Animal Class.

class Baby_Cat(Cat): # Cat Class 상속

_sort_of_animal = '아기 고양이'

```
def __init__(self,name,age,w):
    print("\nb-init) Baby_Cat Class 생성자")
    print('b-init-1) 이름 : ', name)
    print('b-init-2) 나이 : ', age)
    print('b-init-3) 몸무게 : ', w )
    super().__init__(name,age,w)
```

```
def disp(self): # Cat Class의 disp() method 다시 override
    print("\nb-d-1) 이름 : ' + self._name)
    print('b-d-2) 종류 : ' + self._kind_of_animal) #주목 : _sort_of_animal를 사용하지 않고 상위
                                                    #      clsss (CAT)이 멤버변수를 사용

    print('b-d-3) 나이 : ' + str(self._age))
    print('b-d-4) 이름 : ' + str(self._w)+' kg')
    self.sound()
```

```
def sound(self):
    print('b-s-1) 미야요 ~~ 야옹~~')
```

```
0) main에서 3개 객체 생성

t-init) Tiger Class 생성자
t-init-1) name = 호돌이
t-init-2) age = 8
t-init-3) weight = 180

a-init) Animal Class 생성자
a-init-1) name = 호돌이
a-init-2) age = 8
a-init-3) weight = 180

d-init) Dog Class 생성자
d-init-1) name = 멍멍이
d-init-2) age = 4
d-init-3) weight = 8

a-init) Animal Class 생성자
a-init-1) name = 멍멍이
a-init-2) age = 4
a-init-3) weight = 8

c-init) Cat Class 생성자
c-init-1) 이름 : 야옹이
c-init-2) 나이 : 3
c-init-3) 몸무게 : 2

a-init) Animal Class 생성자
a-init-1) name = 야옹이
a-init-2) age = 3
a-init-3) weight = 2
```

```
1) Tiger 클래스 객체 _tiger_obj 호출
t-d-1) 이름 : 호돌이
t-d-2) 종류 : 호랑이
t-d-3) 나이 : 8
t-d-4) 몸무게 : 180kg
t-s-1) 어흥~~
```

```
2) Dog 클래스 객체 _dog_obj 호출
d-d-1) 이름 : 멍멍이
d-d-2) 종류 : 강아지
d-d-3) 나이 : 4
d-d-4) 몸무게 : 8 kg
d-s-1) 멍멍~~
```

3) Cat 클래스 객체 _cat_obj 호출

```
c-d-1) 이름 : 야옹이
c-d-2) 종류 : 고양이
c-d-3) 나이 : 3
c-d-4) 몸무게 : 2 kg
```

c-s-1) 야옹~~

a-ex-1) Animal Class에 대한 설명
a-ex-2) Tiger, Dog, Cat Class의 부모 class입니다.

4) a_cat_obj.copyright_of_Animal_class = @Copyright Animal Class.

5) Baby_Cat 클래스 객체 b_cat 호출

```
b-init) Baby_Cat Class 생성자
b-init-1) 이름 : 나비야
b-init-2) 나이 : 0.2
b-init-3) 몸무게 : 1
```

```
c-init) Cat Class 생성자
c-init-1) 이름 : 나비야
c-init-2) 나이 : 0.2
c-init-3) 몸무게 : 1
```

```
a-init) Animal Class 생성자
a-init-1) name = 나비야
a-init-2) age = 0.2
a-init-3) weight = 1
b-d-1) 이름 : 나비야
b-d-2) 종류 : 고양이
b-d-3) 나이 : 0.2
b-d-4) 이름 : 1 kg
b-s-1) 미야요 ~~ 야옹~~
```

a-ex-1) Animal Class에 대한 설명
a-ex-2) Tiger, Dog, Cat Class의 부모 class입니다.

6) animal_3.copyright_of_Animal_class = @Copyright Animal Class.

<pre> #main print('0) main에서 3개 객체 생성') _tiger_obj = Tiger('호돌이', 8, 180) _dog_obj = Dog('멍멍이', 4, 8) _cat_obj = Cat('야옹이', 3, 2) print('\n1) Tiger 클래스 객체 _tiger_obj 호출') _tiger_obj.disp() print('\n2) Dog 클래스 객체 _dog_obj 호출') _dog_obj.disp() print('\n3) Cat 클래스 객체 _cat_obj 호출') _cat_obj.disp() # 하위 class (_cat_obj) 에서 상위 class(Animal)의 method, attribute 호출 _cat_obj.explain_Animal_Class() print("\n4) a_cat_obj.copyright_of_Animal_class = ", _cat_obj.copyright_of_Animal_class) # Animal > Cat > Baby_Cat class 정의 print('\n5) Baby_Cat 클래스 객체. b_cat 호출') b_cat = Baby_Cat('나비야', 0.2, 1) b_cat.disp() # b_cat 에서 최상위class(Animal)의 method, attribute 호출 b_cat.explain_Animal_Class() print("\n6) animal_3.copyright_of_Animal_class = ", b_cat.copyright_of_Animal_class) </pre>	<pre> 0) main에서 3개 객체 생성 t-init) Tiger Class 생성자 t-init-1) name = 호돌이 t-init-2) age = 8 t-init-3) weight = 180 a-init) Animal Class 생성자 a-init-1) name = 호돌이 a-init-2) age = 8 a-init-3) weight = 180 d-init) Dog Class 생성자 d-init-1) name = 멍멍이 d-init-2) age = 4 d-init-3) weight = 8 a-init) Animal Class 생성자 a-init-1) name = 멍멍이 a-init-2) age = 4 a-init-3) weight = 8 c-init) Cat Class 생성자 c-init-1) 이름 : 야옹이 c-init-2) 나이 : 3 c-init-3) 몸두께 : 2 a-init) Animal Class 생성자 a-init-1) name = 야옹이 a-init-2) age = 3 a-init-3) weight = 2 </pre>
	<pre> 1) Tiger 클래스 객체 _tiger_obj 호출 t-d-1) 이름 : 호돌이 t-d-2) 종류 : 호랑이 t-d-3) 나이 : 8 t-d-4) 몸두께 : 180kg t-s-1) 어흥~~ 2) Dog 클래스 객체 _dog_obj 호출 d-d-1) 이름 : 멍멍이 d-d-2) 종류 : 강아지 d-d-3) 나이 : 4 d-d-4) 몸두께 : 8 kg d-s-1) 멍멍~~ 3) Cat 클래스 객체 _cat_obj 호출 c-d-1) 이름 : 야옹이 c-d-2) 종류 : 고양이 c-d-3) 나이 : 3 c-d-4) 몸두께 : 2 kg c-s-1) 야옹~~ a-ex-1) Animal Class에 대한 설명 a-ex-2) Tiger, Dog, Cat Class의 부모 class입니다. 4) a_cat_obj.copyright_of_Animal_class = @Copyright Animal Class. 5) Baby_Cat 클래스 객체. b_cat 호출 b-init) Baby_Cat Class 생성자 b-init-1) 이름 : 나비야 b-init-2) 나이 : 0.2 b-init-3) 몸두께 : 1 c-init) Cat Class 생성자 c-init-1) 이름 : 나비야 c-init-2) 나이 : 0.2 c-init-3) 몸두께 : 1 a-init) Animal Class 생성자 a-init-1) name = 나비야 a-init-2) age = 0.2 a-init-3) weight = 1 b-d-1) 이름 : 나비야 b-d-2) 종류 : 고양이 b-d-3) 나이 : 0.2 b-d-4) 이름 : 1 kg b-s-1) 미야요 ~~ 야옹~~ a-ex-1) Animal Class에 대한 설명 a-ex-2) Tiger, Dog, Cat Class의 부모 class입니다. 6) animal_3.copyright_of_Animal_class = @Copyright Animal Class. </pre>

0) main에서 3개 객체 생성

t-init) Tiger Class 생성자
t-init-1) name = 호돌이
t-init-2) age = 8
t-init-3) weight = 180

a-init) Animal Class 생성자
a-init-1) name = 호돌이
a-init-2) age = 8
a-init-3) weight = 180

d-init) Dog Class 생성자
d-init-1) name = 멍멍이
d-init-2) age = 4
d-init-3) weight = 8

a-init) Animal Class 생성자
a-init-1) name = 멍멍이
a-init-2) age = 4
a-init-3) weight = 8

c-init) Cat Class 생성자
c-init-1) 이름 : 야옹이
c-init-2) 나이 : 3
c-init-3) 몸두게 : 2

a-init) Animal Class 생성자
a-init-1) name = 야옹이
a-init-2) age = 3
a-init-3) weight = 2

1) Tiger 클래스 객체 _tiger_obj 호출
t-d-1) 이름 : 호돌이
t-d-2) 종류 : 호랑이
t-d-3) 나이 : 8
t-d-4) 몸두게 : 180kg
t-s-1) 어흥~~

2) Dog 클래스 객체 _dog_obj 호출
d-d-1) 이름 : 멍멍이
d-d-2) 종류 : 강아지
d-d-3) 나이 : 4
d-d-4) 몸두게 : 8 kg
d-s-1) 멍멍~~

3) Cat 클래스 객체 _cat_obj 호출
c-d-1) 이름 : 야옹이
c-d-2) 종류 : 고양이
c-d-3) 나이 : 3
c-d-4) 몸두게 : 2 kg
c-s-1) 야옹~~

a-ex-1) Animal Class에 대한 설명

a-ex-2) Tiger, Dog, Cat Class의 부모 class입니다.

4) a_cat_obj.copyright_of_Animal_class = @Copyright Animal Class.

5) Baby_Cat 클래스 객체. b_cat 호출

b-init) Baby_Cat Class 생성자
b-init-1) 이름 : 나비야
b-init-2) 나이 : 0.2
b-init-3) 몸두게 : 1

c-init) Cat Class 생성자
c-init-1) 이름 : 나비야
c-init-2) 나이 : 0.2
c-init-3) 몸두게 : 1

a-init) Animal Class 생성자
a-init-1) name = 나비야
a-init-2) age = 0.2
a-init-3) weight = 1
b-d-1) 이름 : 나비야
b-d-2) 종류 : 고양이
b-d-3) 나이 : 0.2
b-d-4) 이름 : 1 kg
b-s-1) 미야요 ~~ 야옹~~

a-ex-1) Animal Class에 대한 설명

a-ex-2) Tiger, Dog, Cat Class의 부모 class입니다.

6) animal_3.copyright_of_Animal_class = @Copyright Animal Class.

```

class Animal:

    copyright_of_Animal_class = "@Copyright Animal Class."

    def __init__(self, name, age, weight):
        print('\nA-1) Init.')
        self._name = name
        self._age = age
        self._w = weight

    def explain_Animal_Class(self):
        print('\nA-2) Tiger, Dog, Cat Class의 부모 class입니다.')

    def disp(self):
        pass

    def sound(self):
        print("\nA-3) Animal Class : 동물의 울음소리를 출력합니다")

class Cat(Animal): # Animal Class 상속

    _kind_of_animal = '고양이'

    def __init__(self, name, age, w):
        print('\nC-1) Init 1.')
        super().__init__(name, age, w)
        print('\nC-2) Init 2.')

    def disp(self): # Animal Class의 disp() method override
        print('\nC-3) 이름 : ' + self._name)
        print('C-4) 종류 : ' + self._kind_of_animal)
        print('C-5) 나이 : ' + str(self._age))
        print('C-6) 몸무게 : ' + str(self._w) + ' kg')
        self.sound()
        super().sound()

    def sound(self): # Animal Class의 sound() method override
        print('\nC-7) 야옹~~')

```

Super class의 method 호출

1) Cat 클래스 객체 _cat_obj 생성

C-1) Init 1.

A-1) Init.

C-2) Init 2.

C-3) 이름 : 고양이

C-4) 종류 : 고양이

C-5) 나이 : 0.2

C-6) 몸무게 : 1 kg

C-7) 야옹~~

A-3) Animal Class : 동물의 울음소리를 출력합니다

2) Baby_Cat 클래스 객체, b_cat 호출

B-1) Init 1.

C-1) Init 1.

A-1) Init.

C-2) Init 2.

B-2) Init 2.

B-3) 이름 : 나비야

B-4) 종류 : 고양이

B-5) 나이 : 0.2

B-6) 이름 : 1 kg

B-7) 미야요 ~~ 야옹~~

C-7) 야옹~~

A-2) Tiger, Dog, Cat Class의 부모 class입니다.

Super class의 method 호출

```
class Baby_Cat(Cat): # Cat Class 상속
```

```
    _sort_of_animal = '아기 고양이'
```

```
def __init__(self, name, age, w):
    print('\nB-1) Init 1.')
    super().__init__(name, age, w)
    print('\nB-2) Init 2.')
```

```
def disp(self): # Cat Class의 disp() method 다시 override
    print('\nB-3) 이름 : ' + self._name)
    print('\nB-4) 종류 : ' + self._kind_of_animal) # _sort_of_animal를 사용하지 않고
                                                    # 상위 class CAT의 멤버변수를 사용
```

```
    print('\nB=5) 나이 : ' + str(self._age))
    print('\nB=6) 이름 : ' + str(self._w)+' kg')
    self.sound()
    super().sound()
```

```
def sound(self):
    print('\nB-7) 미야요 ~~ 야옹~~')
```

```
#main
print('\n1) Cat 클래스 객체 _cat_obj 생성')
_cat_obj = Cat('고양이', 0.2, 1)
_cat_obj.disp()
```

```
# Animal > Cat > Baby_Cat class 정의
print('\n2) Baby_Cat 클래스 객체, b_cat 호출')
b_cat = Baby_Cat('나비아', 0.2, 1)
b_cat.disp()
b_cat.explain_Animal_Class()
```

1) Cat 클래스 객체 _cat_obj 생성

C-1) Init 1.

A-1) Init.

C-2) Init 2.

C-3) 이름 : 고양이

C-4) 종류 : 고양이

C-5) 나이 : 0.2

C-6) 몸무게 : 1 kg

C-7) 야옹~~

A-3) Animal Class : 동물의 울음소리를 출력합니다

2) Baby_Cat 클래스 객체, b_cat 호출

B-1) Init 1.

C-1) Init 1.

A-1) Init.

C-2) Init 2.

B-2) Init 2.

B-3) 이름 : 나비아

B-4) 종류 : 고양이

B=5) 나이 : 0.2

B=6) 이름 : 1 kg

B-7) 미야요 ~~ 야옹~~

C-7) 야옹~~

A-2) Tiger, Dog, Cat Class의 부모 class입니다.

...

```

class Animal:

    copyright_of_Animal_class = "@Copyright Animal Class."

    def __init__(self, name, age, weight):
        print('\nA-1) Init.')
        self._name = name
        self._age = age
        self._w = weight

    def explain_Animal_Class(self):
        print('\nA-2) Tiger, Dog, Cat Class의 부모 class입니다.')

    def disp(self):
        pass

    def sound(self):
        print("\nA-3) Animal Class : 동물의 울음소리를 출력합니다")

class Cat(Animal): # Animal Class 상속

    _kind_of_animal = '고양이'

    def __init__(self, name, age, w):
        print('\nC-1) Init 1.')
        super().__init__(name, age, w)
        print('\nC-2) Init 2.')

    def disp(self): # Animal Class의 disp() method override
        print('\nC-3) 이름 : ' + self._name)
        print('C-4) 종류 : ' + self._kind_of_animal)
        print('C-5) 나이 : ' + str(self._age))
        print('C-6) 몸두께 : ' + str(self._w) + ' kg')
        self.sound()
        super().sound()

    def sound(self): # Animal Class의 sound() method override
        print('\nC-7) 야옹~~')

```

Super class의 method 호출

1) Cat 클래스 객체 _cat_obj 생성

C-1) Init 1.

A-1) Init.

C-2) Init 2.

C-3) 이름 : 고양이

C-4) 종류 : 고양이

C-5) 나이 : 0.2

C-6) 몸두께 : 1 kg

C-7) 야옹~~

A-3) Animal Class : 동물의 울음소리를 출력합니다

2) Baby_Cat 클래스 객체. b_cat 호출

B-1) Init 1.

C-1) Init 1.

A-1) Init.

C-2) Init 2.

B-2) Init 2.

B-3) 이름 : 나비아

B-4) 종류 : 고양이

B-5) 나이 : 0.2

B-6) 이름 : 1 kg

B-9) 미야요 ~~ 야옹~~

B-7) 부모 class Cat의 sound() method invocation

C-7) 야옹~~

C-7) 야옹~~

B-8) 부모의 부모 class Animal의 sound() method invocation

A-3) Animal Class : 동물의 울음소리를 출력합니다

A-2) Tiger, Dog, Cat Class의 부모 class입니다.

class Baby_Cat(Cat): # Cat Class 상속

_sort_of_animal = '아기 고양이'

```
def __init__(self, name, age, w):
    print('\nB-1) Init 1.')
    super().__init__(name, age, w)
    print('\nB-2) Init 2.')
```

```
def disp(self): # Cat Class의 disp() method 다시 override
    print('\nB-3) 이름 : ' + self._name)
    print('\nB-4) 종류 : ' + self._kind_of_animal) # _sort_of_animal를 사용하지 않고
                                                    # 상위 class CAT의 멤버변수를 사용
```

```
    print('\nB-5) 나이 : ' + str(self._age))
    print('\nB-6) 이름 : ' + str(self._w)+' kg')
```

self.sound()

```
print('\nB-7) 부모 class Cat의 sound() method invocation')
super().sound()
Cat.sound(self)
```

```
print('\nB-8) 부모의 부모 class Animal의 sound() method invocation')
Animal.sound(self)
```

```
def sound(self):
    print('\nB-9) 미야요 ~~ 야옹~~')
```

```
#main
print('\n1) Cat 클래스 객체 _cat_obj 생성')
_cat_obj = Cat('고양이', 0.2, 1)
_cat_obj.disp()
```

```
# Animal > Cat > Baby_Cat class 정의
print('\n2) Baby_Cat 클래스 객체 b_cat 호출')
b_cat = Baby_Cat('나비야', 0.2, 1)
b_cat.disp()
b_cat.explain_Animal_Class()
```

부모의 부모 class method 호출

1) Cat 클래스 객체 _cat_obj 생성

C-1) Init 1.

A-1) Init.

C-2) Init 2.

C-3) 이름 : 고양이

C-4) 종류 : 고양이

C-5) 나이 : 0.2

C-6) 몸무게 : 1 kg

C-7) 야옹~~

A-3) Animal Class : 동물의 울음소리를 출력합니다

2) Baby_Cat 클래스 객체 b_cat 호출

B-1) Init 1.

C-1) Init 1.

A-1) Init.

C-2) Init 2.

B-2) Init 2.

B-3) 이름 : 나비야

B-4) 종류 : 고양이

B-5) 나이 : 0.2

B-6) 이름 : 1 kg

B-9) 미야요 ~~ 야옹~~

B-7) 부모 class Cat의 sound() method invocation

C-7) 야옹~~

C-7) 야옹~~

B-8) 부모의 부모 class Animal의 sound() method invocation

A-3) Animal Class : 동물의 울음소리를 출력합니다

A-2) Tiger, Dog, Cat Class의 부모 class입니다.

super를 사용하지 않고 자식 class에서 부모 class를 초기화

```
inheritance_no_super.py
File Edit Format Run Options Window Help
1 # super를 사용하지 않고 자식 클래스에서 부모 클래스를 초기화
2
3 class Parent:
4
5     def __init__(self, value):
6         print('Wnp-1) Init.')
7         self.value_in_Parent = value
8         print('p-2) self.value_in_Parent = ', self.value_in_Parent)
9
10
11 class Child(Parent):
12
13     def __init__(self, value):
14         print('Wnc-1) Init.')
15         Parent.__init__(self, value)
16         print('Wnc-2) self.value_in_Parent = ', self.value_in_Parent)
17
18
19 #main
20 print('1) main')
21 kids = Child(5)
22 print('2) kids.value_in_Parent = ', kids.value_in_Parent)
23
```

1) main

c-1) Init.

p-1) Init.

p-2) self.value_in_Parent = 5

c-2) self.value_in_Parent = 5

2) kids.value_in_Parent = 5

>>>

```

1 class Parent_Park:
2
3     def __init__(self, value):
4         print('PARK-1) Init.')
5         self.in_class_value = value
6         print('PARK-2) self.in_class_value -> ', self.in_class_value)
7
8
9 class Parent_KIM:
10
11     def __init__(self):
12         print('\nKIM-1) Init.')
13         self.in_class_value *= 2
14         print('KIM-2) self.in_class_value * 2 -> ', self.in_class_value)
15
16
17 class Parent_LEE:
18
19     def __init__(self):
20         print('\nLEE-1) Init.')
21         self.in_class_value += 5
22         print('LEE-2) self.in_class_value + 5 -> ', self.in_class_value, '\n')
23
24
25 class Child(Parent_Park, Parent_KIM, Parent_LEE):
26
27     def __init__(self, value):
28         print('\nChild) Init.\n')
29         Parent_Park.__init__(self, value)
30         Parent_KIM.__init__(self)
31         Parent_LEE.__init__(self)
32
33 #main
34 print('1) main')
35 kids = Child(5)
36 print('2) (5*2) + 5 = ', kids.in_class_value)

```

Multiple Inheritance -1

: __init__

```

1) main
Child) Init.

PARK-1) Init.
PARK-2) self.in_class_value -> 5

KIM-1) Init.
KIM-2) self.in_class_value * 2 -> 10

LEE-1) Init.
LEE-2) self.in_class_value + 5 -> 15

2) (5*2) + 5 = 15

```

```

class Parent_Park:
    def __init__(self, value):
        print('PARK-1) Init.')
        self.in_class_value = value
        print('PARK-2) self.in_class_value -> ', self.in_class_value)

    def disp(self):
        number_of_PARKs_Family = 50
        print('\nPARK-3) number_of_PARKs_Family = ', number_of_PARKs_Family)

class Parent_KIM:
    def __init__(self):
        print('\nKIM-1) Init.')
        self.in_class_value += 2
        print('KIM-2) self.in_class_value + 2 -> ', self.in_class_value)

    def disp(self):
        number_of_KIMs_Family = 100
        print('\nKIM-3) number_of_KIMs_Family = ', number_of_KIMs_Family)

class Parent_LEE:
    def __init__(self):
        print('\nLEE-1) Init.')
        self.in_class_value += 5
        print('LEE-2) self.in_class_value + 5 -> ', self.in_class_value, '\n')

    def disp(self):
        number_of_LEEs_Family = 200
        print('\nLEE-3) number_of_LEEs_Family = ', number_of_LEEs_Family)

순서!
class Child(Parent_Park, Parent_KIM, Parent_LEE):
    def __init__(self, value):
        print('Child) Init.\n')
        Parent_Park.__init__(self, value)
        Parent_KIM.__init__(self)
        Parent_LEE.__init__(self)

```

Multiple Inheritance -2

: method

```

1) main
Child) Init.

PARK-1) Init.
PARK-2) self.in_class_value -> 5

KIM-1) Init.
KIM-2) self.in_class_value + 2 -> 10

LEE-1) Init.
LEE-2) self.in_class_value + 5 -> 15

2) (5+2) + 5 = 15

3) 과연 어떤 부모 클래스의 disp() method가 불리울까 ?

PARK-3) number_of_PARKs_Family = 50

```

Multiple Inheritance -2

: method

```
#main
print('1) main')
kids = Child(5)
print('2) (5*2) + 5) = ', kids.in_class_value)

print('3) 과연 어떤 부모 클래스의 disp() method가 불리울까 ?')
kids.disp()
```

```
1) main
Child) Init.

PARK-1) Init.
PARK-2) self.in_class_value -> 5

KIM-1) Init.
KIM-2) self.in_class_value + 2 -> 10

LEE-1) Init.
LEE-2) self.in_class_value + 5 -> 15

2) (5*2) + 5) = 15

3) 과연 어떤 부모 클래스의 disp() method가 불리울까 ?

PARK-3) number_of_PARKs_Family = 50
```

Multiple Inheritance -2 : method

```
...  
class Child(Parent_Park, Parent_KIM, Parent_LEE):  
    def __init__(self, value):  
        print('Child) Init. #n')  
        Parent_Park.__init__(self, value)  
        Parent_KIM.__init__(self)  
        Parent_LEE.__init__(self)  
...
```

```
class Child(Parent_KIM, Parent_Park, Parent_LEE):  
    def __init__(self, value):  
        print('Child) Init. #n')  
        Parent_Park.__init__(self, value)  
        Parent_KIM.__init__(self)  
        Parent_LEE.__init__(self)  
...
```

```
class Child(Parent_LEE, Parent_KIM, Parent_Park):  
    def __init__(self, value):  
        print('Child) Init. #n')  
        Parent_Park.__init__(self, value)  
        Parent_KIM.__init__(self)  
        Parent_LEE.__init__(self)  
...
```

1) main
Child) Init.

PARK-1) Init.
PARK-2) self.in_class_value -> 5

KIM-1) Init.
KIM-2) self.in_class_value * 2 -> 10

LEE-1) Init.
LEE-2) self.in_class_value + 5 -> 15

2) $(5*2) + 5 = 15$

3) 과연 어떤 부모 클래스의 disp() method가 불리울까 ?

KIM-3) number_of_KIMs_Family = 100

Multiple Inheritance -2 : method

```
...
class Child(Parent_Park, Parent_KIM, Parent_LEE):
    def __init__(self, value):
        print('Child) Init.\n')
        Parent_Park.__init__(self, value)
        Parent_KIM.__init__(self)
        Parent_LEE.__init__(self)

class Child(Parent_KIM, Parent_Park, Parent_LEE):
    def __init__(self, value):
        print('Child) Init.\n')
        Parent_Park.__init__(self, value)
        Parent_KIM.__init__(self)
        Parent_LEE.__init__(self)
...

class Child(Parent_LEE, Parent_KIM, Parent_Park):
    def __init__(self, value):
        print('Child) Init.\n')
        Parent_Park.__init__(self, value)
        Parent_KIM.__init__(self)
        Parent_LEE.__init__(self)
```

```
1) main
Child) Init.

PARK-1) Init.
PARK-2) self.in_class_value -> 5

KIM-1) Init.
KIM-2) self.in_class_value + 2 -> 10

LEE-1) Init.
LEE-2) self.in_class_value + 5 -> 15
```

2) $(5+2) + 5 = 15$

3) 과연 어떤 부모 클래스의 disp() method가 불리울까 ?

LEE-3) number_of_LEEs_Family = 200

MRO : Method Resolution Order

- 기저 클래스(parent classes)의 지정 순서를 바꾸면 호출되는 method도 변화
 - 이는 Python의 **method 해석순서(MRO : Method Resolution Order)**에 정해진 규칙에 따라 수행
 - Class object의 `__mro__` 속성을 참고하면 알 수 있음
 - 예를 들어 변경전의 클래스의
 - `Child.__mro__` 또는
 - `Child.mro()`
- 속성 값은 다음 페이지와 같음


```
class Child(Parent_Park, Parent_KIM, Parent_LEE):
```

```
    def __init__(self, value):  
        print('Child) Init.\\n')  
        Parent_Park.__init__(self, value)  
        Parent_KIM.__init__(self)  
        Parent_LEE.__init__(self)
```

```
...
```

```
class Child(Parent_KIM, Parent_Park, Parent_LEE):
```

```
    def __init__(self, value):  
        print('Child) Init.\\n')  
        Parent_Park.__init__(self, value)  
        Parent_KIM.__init__(self)  
        Parent_LEE.__init__(self)
```

```
class Child(Parent_LEE, Parent_KIM, Parent_Park):
```

```
    def __init__(self, value):  
        print('Child) Init.\\n')  
        Parent_Park.__init__(self, value)  
        Parent_KIM.__init__(self)  
        Parent_LEE.__init__(self)
```

```
#main
```

```
print('1) main')
```

```
kids = Child(5)
```

```
print('2) (5*2) + 5 = ', kids.in_class_value)
```

```
print('\\n3) 과연 어떤 부모 클래스의 disp() method가 불리울까 ?')
```

```
kids.disp()
```

3) 과연 어떤 부모 클래스의 disp() method가 불리울까 ?

PARK-3) number_of_PARKs_Family = 50

```
>>>
```

```
>>> Child.__mro__
```

```
(<class '__main__.Child'>, <class '__main__.Parent_Park'>, <class '__main__.Parent_KIM'>, <class '__main__.Parent_LEE'>, <class 'object'>)
```

```
>>> Child.mro()
```

```
[<class '__main__.Child'>, <class '__main__.Parent_Park'>, <class '__main__.Parent_KIM'>, <class '__main__.Parent_LEE'>, <class 'object'>]
```

```
...
```

```

...
class Child(Parent_Park, Parent_KIM, Parent_LEE):
    def __init__(self, value):
        print('Child) Init.\n')
        Parent_Park.__init__(self, value)
        Parent_KIM.__init__(self)
        Parent_LEE.__init__(self)
...
class Child(Parent_KIM, Parent_Park, Parent_LEE):
    def __init__(self, value):
        print('Child) Init.\n')
        Parent_Park.__init__(self, value)
        Parent_KIM.__init__(self)
        Parent_LEE.__init__(self)
...
class Child(Parent_LEE, Parent_KIM, Parent_Park):
    def __init__(self, value):
        print('Child) Init.\n')
        Parent_Park.__init__(self, value)
        Parent_KIM.__init__(self)
        Parent_LEE.__init__(self)
...

```

3) 과연 어떤 부모 클래스의 disp() method가 불리울까 ?

```

KIM-3) number_of_KIMs_Family = 100
>>> Child.__mro__
(<class '__main__.Child'>, <class '__main__.Parent_KIM'>, <class '__main__.Parent_Park'>, <class '__main__.Parent_LEE'>, <class 'object'>)
>>> Child.mro()
[<class '__main__.Child'>, <class '__main__.Parent_KIM'>, <class '__main__.Parent_Park'>, <class '__main__.Parent_LEE'>, <class 'object'>]
>>>

```

```

...
class Child(Parent_Park, Parent_KIM, Parent_LEE):

    def __init__(self, value):
        print('Child) Init.\\n')
        Parent_Park.__init__(self, value)
        Parent_KIM.__init__(self)
        Parent_LEE.__init__(self)

class Child(Parent_KIM, Parent_Park, Parent_LEE):

    def __init__(self, value):
        print('Child) Init.\\n')
        Parent_Park.__init__(self, value)
        Parent_KIM.__init__(self)
        Parent_LEE.__init__(self)

...
class Child(Parent_LEE, Parent_KIM, Parent_Park):

    def __init__(self, value):
        print('Child) Init.\\n')
        Parent_Park.__init__(self, value)
        Parent_KIM.__init__(self)
        Parent_LEE.__init__(self)

```

3) 과연 어떤 부모 클래스의 disp() method가 불리울까 ?

```

PARK-3) number_of_LEEs_Family = 200
>>> Child.__mro__
(<class '__main__.Child'>, <class '__main__.Parent_LEE'>, <class '__main__.Parent_KIM'>, <class '__main__.Parent_Park'>, <class 'object'>)
>>> Child.mro()
[<class '__main__.Child'>, <class '__main__.Parent_LEE'>, <class '__main__.Parent_KIM'>, <class '__main__.Parent_Park'>, <class 'object'>]
>>>

```

다형성 (polymorphism)

- 다형성(polymorphism)은 “많은(poly)+모양(morph)”
- 서로 다른 class에 이름이 같은 여러 개의 method를 둘 수 있음
- 동일한 method에 대하여 구체적인 instance마다 다른 동작을 수행하는 특징.

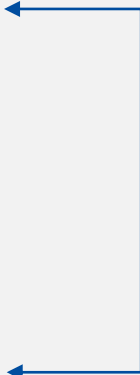
다형성(Polymorphism)



다형성 (polymorphism)

```
class Triangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height
    def getArea(self):
        area = self.width * self.height / 2.0
        return area

class Square:
    def __init__(self, size):
        self.size = size
    def getArea(self):
        area = self.size * self.size
        return area
```



둘 다 getArea()라는
method를 갖고 있지만
각 도형마다 다른 동작을 수행

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())
```

내장 함수와 다형성

```
>>> list = [1, 2, 3]          # 리스트
>>> len(list)
3

>>> s = "This is a sentence"  # 문자열
>>> len(s)
18

>>> d = {'aaa': 1, 'bbb': 2}   # 딕셔너리
>>> len(d)
2
```


Lab: Vehicle와 Car, Truck

- 일반적인 운송수단을 나타내는 Vehicle Class를 상속받아서 Car Class와 Truck Class를 작성해보자.

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

Solution

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

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

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

Solution

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

    def stop(self):
        return '승용차를 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())
```

정적 method(static method)와 Class method(class method)

- 정적 method : 객체를 이용하지 않고 class 이름을 통하여 직접 호출할 수 있는 method
 - 일반 method와는 달리 첫 인수로 self를 받지 않으며 필요한 만큼의 인수를 선언하여 사용
 - staticmethod()를 사용해 정적 method 등록
<호출할 method 이름> = staticmethod(class 내에 정의한 method 이름>
- Class method : instance에서 정적 method 처럼 사용할 수 있도록 만들어 줌
 - 암묵적으로 객체의 class가 매개변수로 넘어감
<호출할 method 이름> = classmethod(class 내에 정의한 method 이름>

인스턴스 객체를 이용하지 않고 클래스 이름을 통하여 직접 호출할 수 있는 메서드인
staticmethod 등록 방법 - TypeError 발생의 경우

```
class CounterManager:

    instance_Count = 0

    def __init__(self): #인스턴스가 생성되면 클래스 영역의 instance_Count 값이 1씩 증가
        CounterManager.instance_Count += 1
        print("5) __init__, CounterManager.instance_Count = ", CounterManager.instance_Count)

    def print_Instance_Count(): # static method 선언, 첫 인수로 self를 받지 않으며
        # 필요한 만큼의 인수를 선언하여 사용
        print("6) In print_Instance_Count(), instance_Count =", CounterManager.instance_Count)

#main
print("1) In Main, 객체 a,b,c 생성")
a, b, c = CounterManager(), CounterManager(), CounterManager()

print("2) 객체 a,b,c 생성 완료")
print("3) staticmethod CounterManager.print_Instance_Count() 호출")
CounterManager.print_Instance_Count()

print("4) staticmethod print_Instance_Count()를 CounterManager class를 통해 호출하지 않고")
print("    객체 b를 통해 호출 --> TypeError 발생")
b.print_Instance_Count()
```

```
1) In Main, 객체 a,b,c 생성
5) __init__, CounterManager.instance_Count = 1
5) __init__, CounterManager.instance_Count = 2
5) __init__, CounterManager.instance_Count = 3
2) 객체 a,b,c 생성 완료
3) staticmethod CounterManager.print_Instance_Count() 호출
6) In print_Instance_Count(), instance_Count = 3
4) staticmethod print_Instance_Count()를 CounterManager class를 통해 호출하지 않
고
   객체 b를 통해 호출 --> TypeError 발생
```

```
Traceback (most recent call last):
  File "C:\Users\박수현\Desktop\2017-1학기 강의\소프트웨어학부\2017과학과소프트
웨어적사고\교안\강의ppt\헬로-파이썬 프로그래밍\static_class_method_14_1.py", lin
e 27, in <module>
    b.print_Instance_Count()
TypeError: print_Instance_Count() takes 0 positional arguments but 1 was given
>>> |
```

인스턴스 객체를 이용하지 않고 클래스 이름을 통하여 직접 호출할 수 있는 메서드인
staticmethod 등록 방법. staticmethod로 등록하여 실행

```
class CounterManager:
    instance_Count = 0

    def __init__(self): #인스턴스가 생성되면 클래스 영역의 instance_Count 값이 1씩 증가
        CounterManager.instance_Count += 1
        print("5) __init__, CounterManager.instance_Count = ", CounterManager.instance_Count)

    def print_Instance_Count(): # staticmethod 정의. 첫 인수로 self를 받지 않으며
        # 필요한 만큼의 인수를 선언하여 사용
        print("6) In print_Instance_Count(), instance_Count = ", CounterManager.instance_Count)

    print("7) print_Instance_Count() method를 staticmethod Static_print_Instance_Count로 등록")
    Static_print_instance_Count = staticmethod(print_Instance_Count)

#main
print("1) In Main, 객체 a,b,c 생성")
a, b, c = CounterManager(), CounterManager(), CounterManager()

print("2) 객체 a,b,c 생성 완료")
print("3) staticmethod CounterManager.print_Instance_Count() 호출")
CounterManager.print_Instance_Count()

print("4) staticmethod print_Instance_Count()를 Static_print_Instance_Count로 호출")
b.Static_print_instance_Count()

7) print_Instance_Count() method를 staticmethod Static_print_Instance_Count로 등록
1) In Main, 객체 a,b,c 생성
5) __init__, CounterManager.instance_Count = 1
5) __init__, CounterManager.instance_Count = 2
5) __init__, CounterManager.instance_Count = 3
2) 객체 a,b,c 생성 완료
3) staticmethod CounterManager.print_Instance_Count() 호출
6) In print_Instance_Count(), instance_Count = 3
4) staticmethod print_Instance_Count()를 Static_print_Instance_Count로 호출
6) In print_Instance_Count(), instance_Count = 3
>>> |
```

```

# classmethod 정의, 인스턴스에서 정적 메소드처럼 사용할 수 있도록 만들어 줌
# 암묵적으로 인스턴스의 클래스가 매개변수로 넘어감

class CounterManager:
    instance_Count = 0

    def __init__(self): #인스턴스가 생성되면 클래스 영역의 instance_Count 값이 1씩 증가
        CounterManager.instance_Count += 1
        print("9) __init__, CounterManager.instance_Count = ", CounterManager.instance_Count)

    def print_Instance_Count(): # staticmethod 정의, 첫 인수로 self를 받지 않으며
        # 필요한 만큼의 인수를 선언하여 사용
        print("10) In print_Instance_Count(), instance_Count =", CounterManager.instance_Count)

    print("11) print_Instance_Count() method를 스테틱메소드 Static_print_Instance_Count로 등록")
    Static_print_instance_Count = staticmethod(print_Instance_Count)

    def print_class_Count(cls): # classmethod 정의, 인스턴스에서 정적 메소드처럼 사용할 수
        # 있도록 만들어 줌
        # 암묵적으로 인스턴스의 클래스가 매개변수로 넘어감
        print("12) In print_class_Count(), instance_Count =", CounterManager.instance_Count)

    print("13) print_class_Count() method를 클래스메소드 Class_print_Instance_Count로 등록")
    Class_print_Instance_Count = classmethod(print_class_Count)

#main
print("1) In Main, 객체 a,b,c 생성")
a, b, c = CounterManager(), CounterManager(), CounterManager()

print("2) 객체 a,b,c 생성 완료")

```

```

# 정적 메소드로 출력
print("3) 정적메소드 CounterManager.print_Instance_Count() 호출")
CounterManager.print_Instance_Count()

print("5) 정적메소드 CounterManager.Static_print_Instance_Count 호출")
CounterManager.Static_print_Instance_Count()

print("6) staticmethod print_Instance_Count()를 정적 메소드 Static_print_Instance_Count로 호출")
b.Static_print_Instance_Count()

# 클래스 메소드로 출력
print("7) 클래스메소드 CounterManager.print_class_Count() 호출")
CounterManager.Class_print_Instance_Count()

print("8) classmethod print_class_Count()를 클래스 메소드 Class_print_Instance_Count로 호출")
b.Class_print_Instance_Count()

```

```

11) print_Instance_Count() method를 스테틱메소드 Static_print_Instance_Count로 등록
13) print_class_Count() method를 클래스메소드 Class_print_Instance_Count로 등록
1) In Main, 객체 a,b,c 생성
9) __init__, CounterManager.instance_Count = 1
9) __init__, CounterManager.instance_Count = 2
9) __init__, CounterManager.instance_Count = 3
2) 객체 a,b,c 생성 완료
3) 정적메소드 CounterManager.print_Instance_Count() 호출
10) In print_Instance_Count(), instance_Count = 3
5) 정적메소드 CounterManager.Static_print_Instance_Count 호출
10) In print_Instance_Count(), instance_Count = 3
6) staticmethod print_Instance_Count()를 정적 메소드 Static_print_Instance_Count로 호출
10) In print_Instance_Count(), instance_Count = 3
7) 클래스메소드 CounterManager.print_class_Count() 호출
12) In print_class_Count(), instance_Count = 3
8) classmethod print_class_Count()를 클래스 메소드 Class_print_Instance_Count로 호출
12) In print_class_Count(), instance_Count = 3
>>> |

```

static_class_method_14_3.py (2/2)

정적 method(static method)와 Class method(class method)

```
class CounterManager:
    insCount = 0

    def __init__(self):
        CounterManager.insCount += 1

    def printInstaceCount():
        print("Instance Count: ", CounterManager.insCount)

#main
a, b, c = CounterManager(), CounterManager(), CounterManager()
CounterManager.printInstaceCount()
b.printInstaceCount()
```

← 인스턴스가 생성되면 Class 영역의 insCount 변수 값이 증가

← "Instance Count: 3"이 출력
Class(CounterManager)를 통해 호출하는 경우는 정
상적으로 수행(정적 method)되나 instance 객체
(b)를 이용하여 호출하는 경우

정적 method(static method)와 Class method(class method)

```
class CounterManager:
    insCount = 0
    def __init__(self):
        CounterManager.insCount += 1
    def staticPrintCount(): ← 정적 method 정의
        print("Instance Count: ", CounterManager.insCount)
    SPrintCount = staticmethod(staticPrintCount) ← 정적 method로 등록

    def classPrintCount(cls): ← Class method 정의
        print("Instance Count: ", CounterManager.insCount)
    CPrintCount = classmethod(classPrintCount) ← Class method로 등록

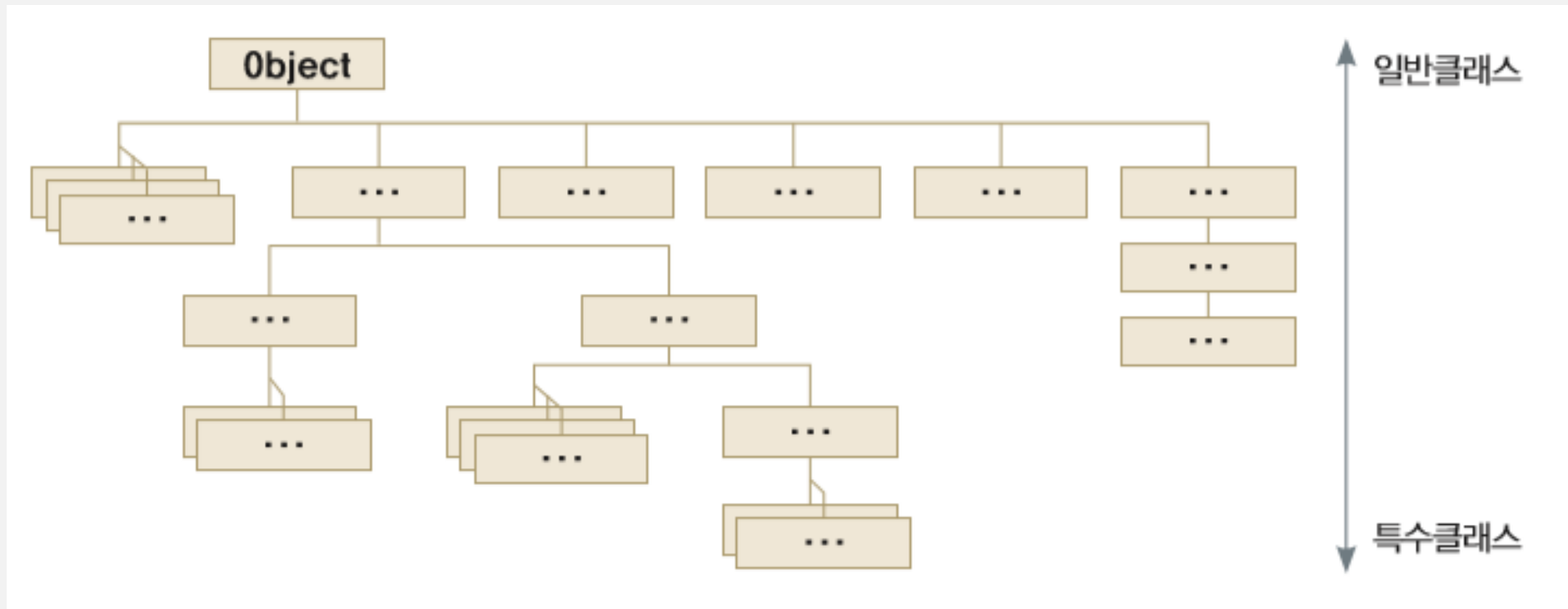
a, b, c = CounterManager(), CounterManager(), CounterManager()

CounterManager.SPrintCount() ← 정적 method로 출력
b.SPrintCount()

CounterManager.CPrintCount() ← Class method로 출력
b.CPrintCount()
```

Object Class

- 모든 Class의 맨 위에는 object Class가 있다고 생각하면 된다.



Object Class의 Method

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)`

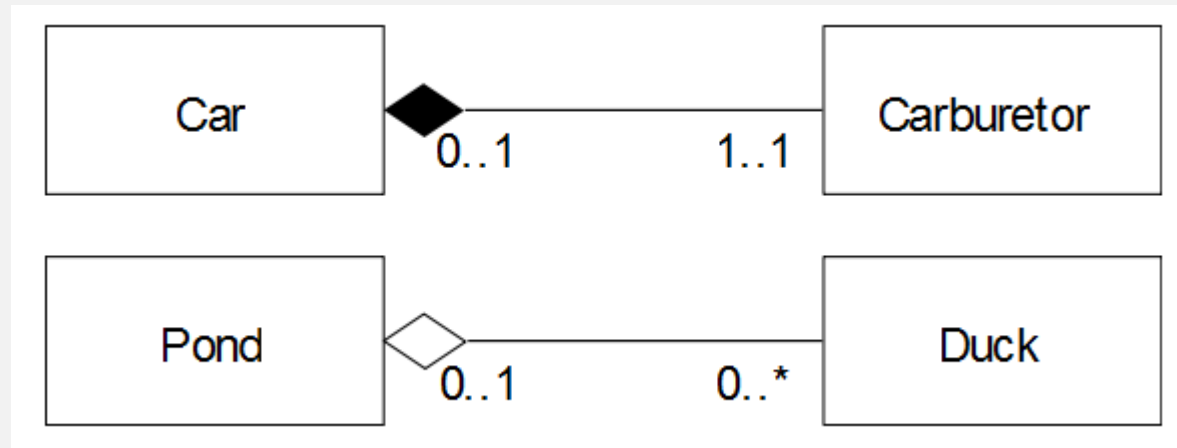
```
class Book:
    def __init__(self, title, isbn):
        self.__title = title
        self.__isbn = isbn
    def __repr__(self):
        return "ISBN: "+ self.__isbn+ "; TITLE: "+ self.__title

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

ISBN: 0123456; TITLE: The Python Tutorial

Class 관계

- is-a 관계: 상속
- has-a 관계



```
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
```

Lab: Card와 Deck

- 카드를 나타내는 Card Class를 작성하고 52개의 Card 객체를 가지고 있는 Deck Class를 작성한다. 각 Class의 `__str__()` Method를 구현하여서 덱 안에 들어 있는 카드를 다음과 같이 출력한다.

```
['클럽 에이스', '클럽 2', '클럽 3', '클럽 4', '클럽 5', '클럽 6', '클럽 7', '클럽 8', '클럽 9', '클럽 10', '클럽 잭', '클럽 퀸', '클럽 킹', '다이아몬드 에이스', '다이아몬드 2', '다이아몬드 3', '다이아몬드 4', '다이아몬드 5', '다이아몬드 6', '다이아몬드 7', '다이아몬드 8', '다이아몬드 9', '다이아몬드 10', '다이아몬드 잭', '다이아몬드 퀸', '다이아몬드 킹', '하트 에이스', '하트 2', '하트 3', '하트 4', '하트 5', '하트 6', '하트 7', '하트 8', '하트 9', '하트 10', '하트 잭', '하트 퀸', '하트 킹', '스페이드 에이스', '스페이드 2', '스페이드 3', '스페이드 4', '스페이드 5', '스페이드 6', '스페이드 7', '스페이드 8', '스페이드 9', '스페이드 10', '스페이드 잭', '스페이드 퀸', '스페이드 킹']
```


Solution

```
class Card:
    suitNames = ['클럽', '다이아몬드', '하트', '스페이드']
    rankNames = [None, '에이스', '2', '3', '4', '5', '6', '7',
                  '8', '9', '10', '잭', '퀸', '킹']

    def __init__(self, suit, rank):
        self.suit = suit
        self.rank = rank

    def __str__(self):
        return Card.suitNames[self.suit]+" "+\
               Card.rankNames[self.rank]
```

Solution

```
class Deck:
    def __init__(self):
        self.cards = []

        for suit in range(4):
            for rank in range(1, 14):
                card = Card(suit, rank)
                self.cards.append(card)
    def __str__(self):
        lst = [str(card) for card in self.cards]
        return str(lst)

deck = Deck()          # 덱 객체를 생성한다.
print(deck)            # 덱 객체를 출력한다. __str__()이 호출된다.
```

핵심 정리

- 상속은 다른 Class를 재사용하는 탁월한 방법이다. 객체와 객체간의 is-a 관계가 성립된다면 상속을 이용하도록 하자.
- 상속을 사용하면 중복된 코드를 줄일 수 있다. 공통적인 코드는 부모 Class를 작성하여 한 곳으로 모으도록 하자.
- 상속에서는 부모 Class의 Method를 자식 Class가 재정의할 수 있다. 이것을 Method Overriding이라고 한다.

Q & A

