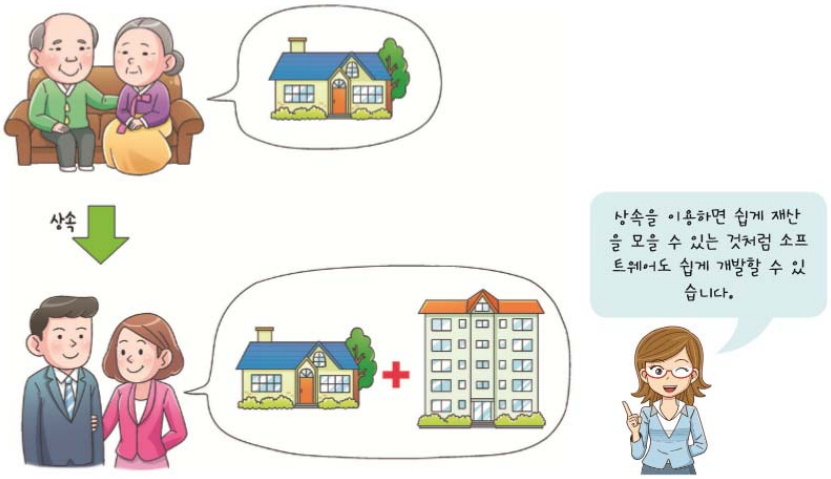
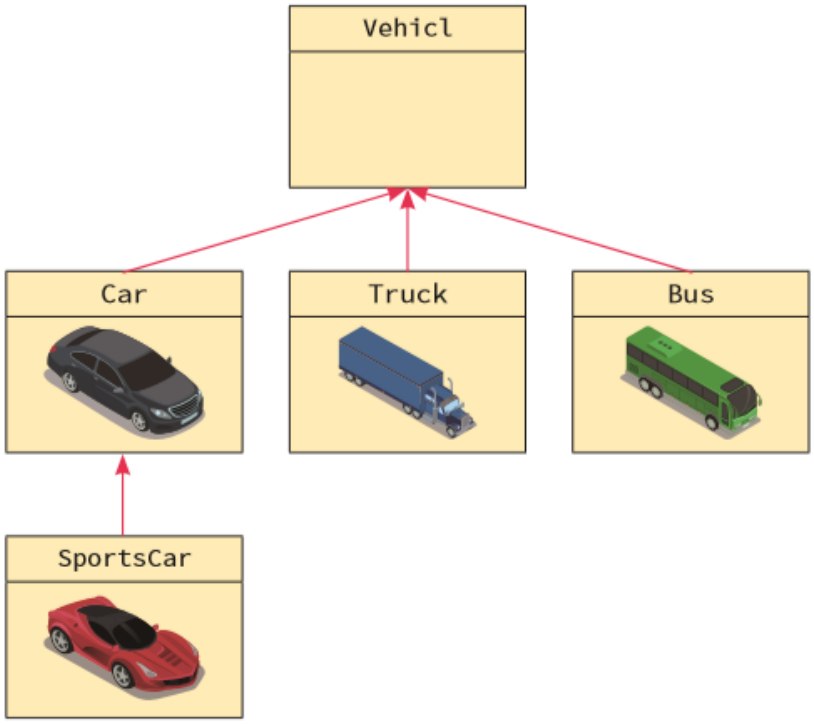
11장. 상속과 다형성

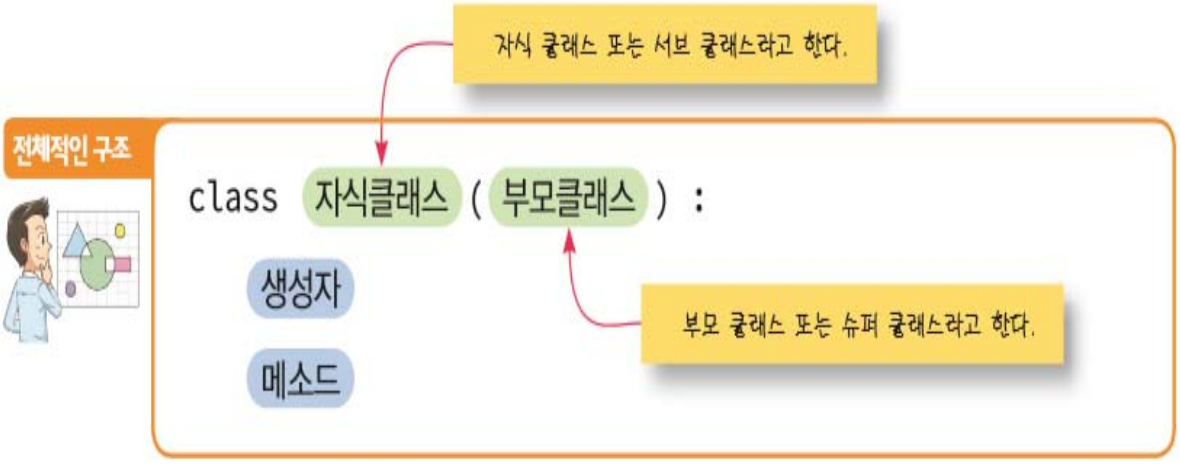
- 상속(inheritance)은 기존에 존재하는 클래스로부터 코드와 데이터를 이어받고 자신이 필요한 기능을 추가하는 기법이다.



상속의 예


Vehicl
Car
Truck
Bus
SportsCar

상속 구현하기

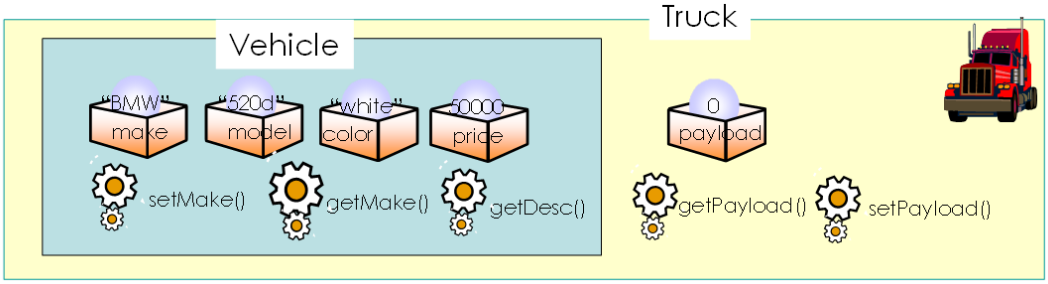

자식 클래스 또는 서브 클래스라고 한다.
전체적인 구조
class 자식클래스 (부모클래스) :
생성자
메소드
부모 클래스 또는 슈퍼 클래스라고 한다.

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)+")"
```

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
```

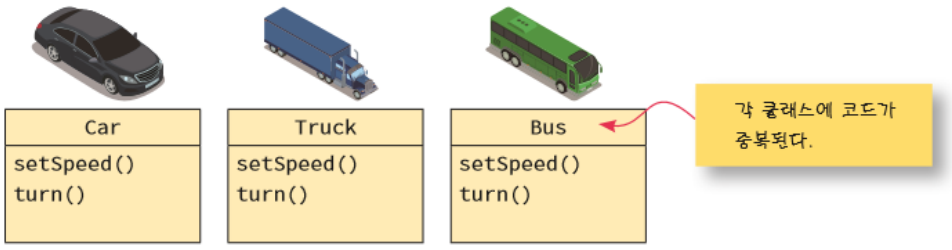


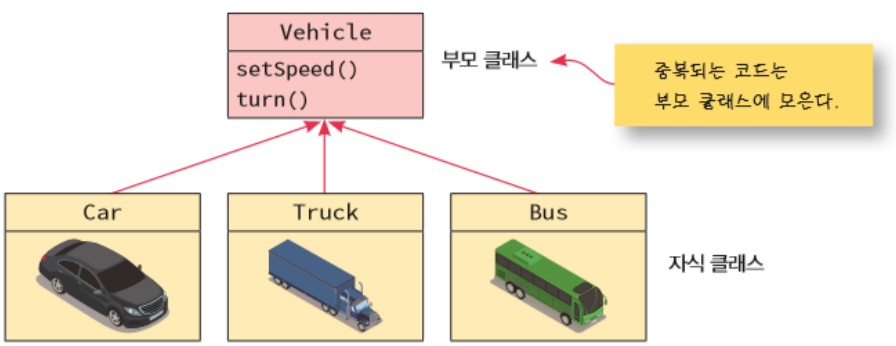
예제

```
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)

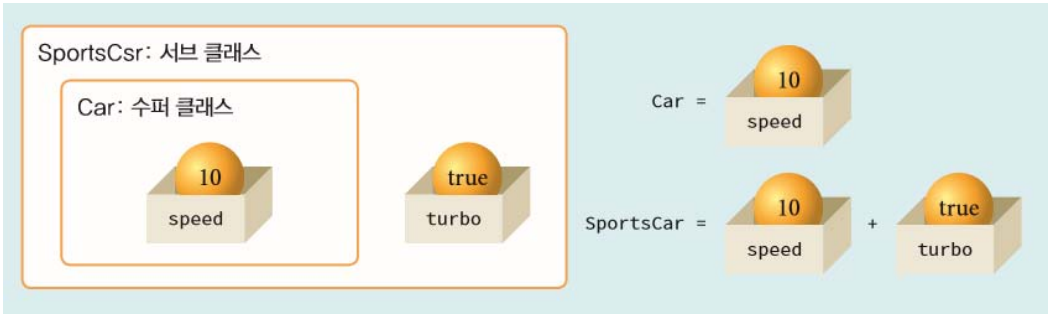
왜 상속을 사용하는가?





Lab: Sportscar 클래스

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

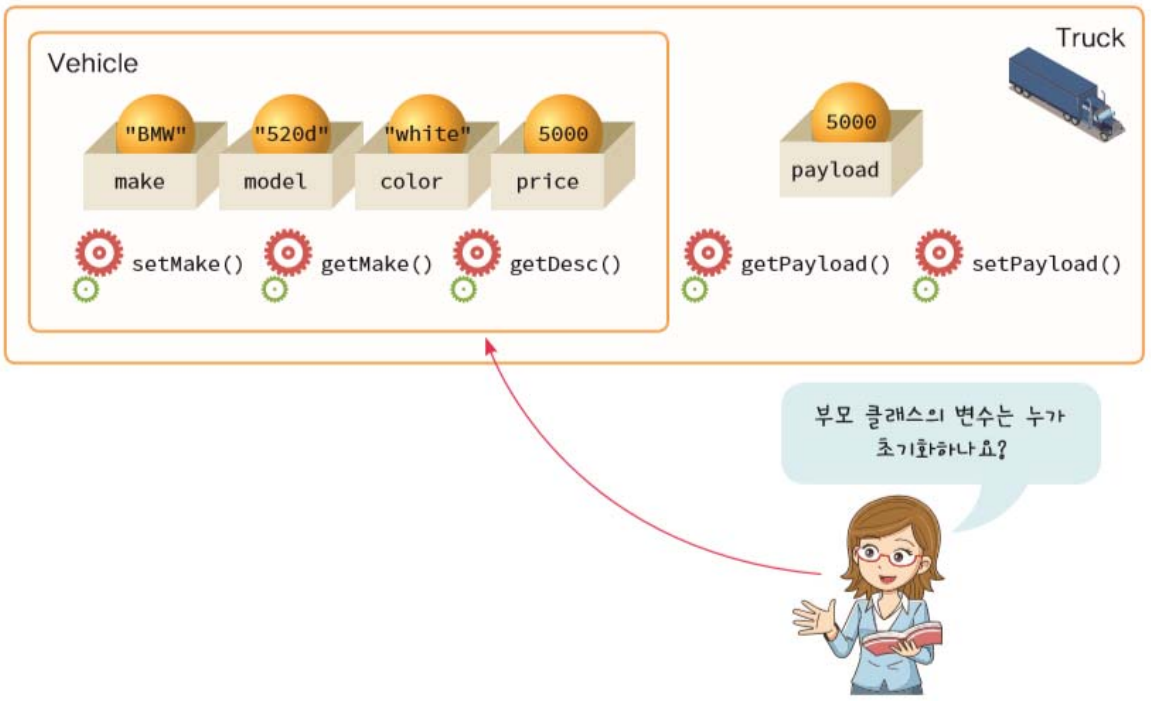
Solution

```
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)
```

Solution

```
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)
```

부모 클래스의 생성자 호출

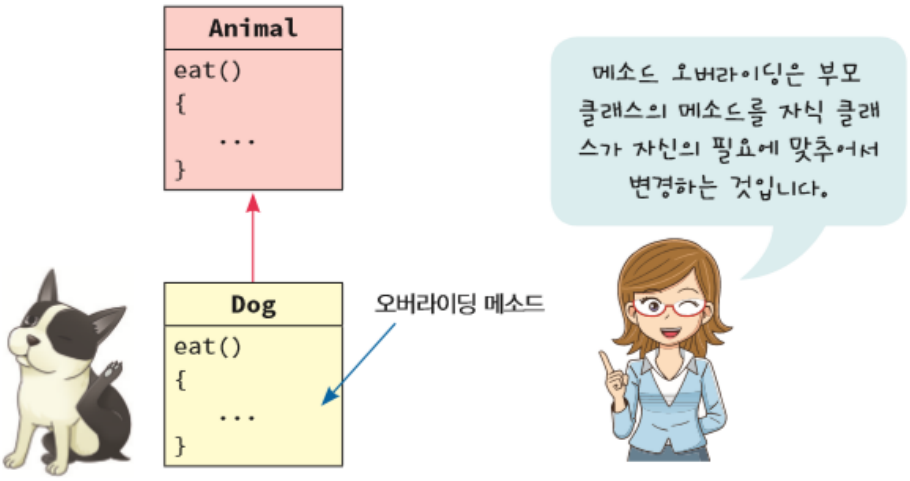


부모 클래스의 생성자를 명시적으로 호출한다.

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

메소드 오버라이딩

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



예제

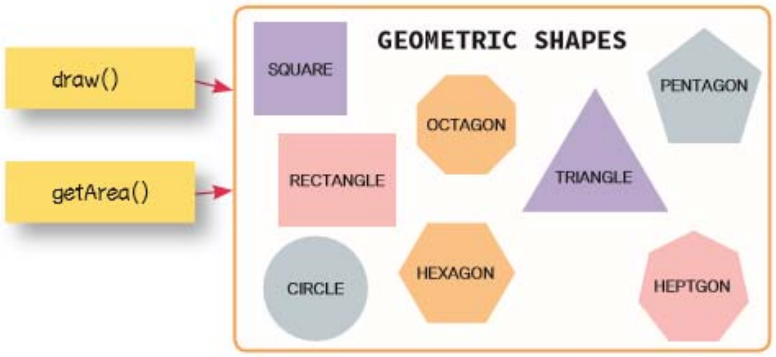
```
class Animal:
    def __init__(self, name=""):
        self.name=name
    def eat(self):
        print("동물이 먹고 있습니다. ")

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

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

강아지가 먹고 있습니다.

다형성

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

다형성의 예



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
```

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())
```

Lab: Vehicle와 Car, Truck

- 일반적인 운송수단을 나타내는 Vehicle 클래스를 상속받아서 Car 클래스와 Truck 클래스를 작성해보자.

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

Solution

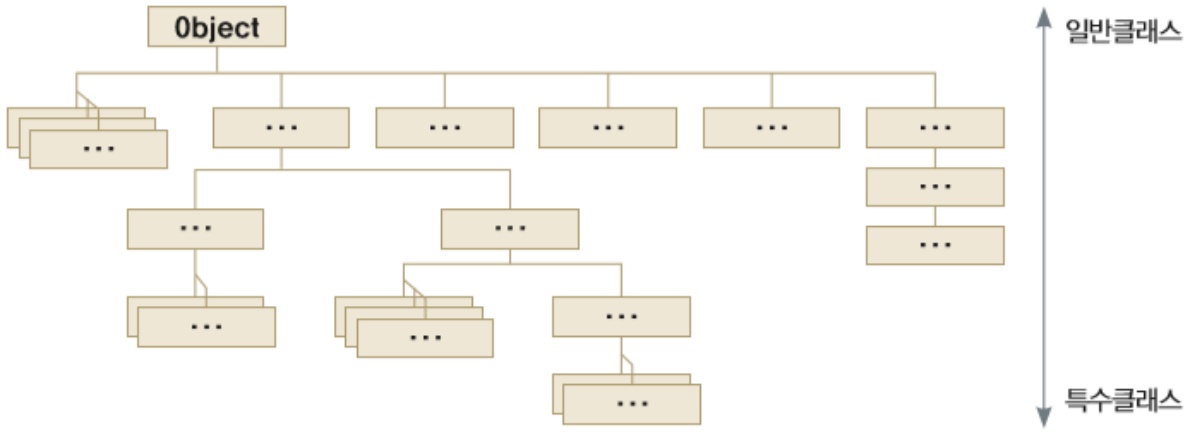
```
class Vehicle:
    def __init__(self, name):
        self.name = name

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

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

Solution

```
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())
```

Object 클래스

- 모든 클래스의 맨 위에는 object 클래스가 있다고 생각하면 된다.



Object 클래스의 메소드

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

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

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.