

Object Oriented Programming

■ 상속이란?

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

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

전체적인 구조



```
class 자식클래스 ( 부모클래스 ) :
```

생성자

메소드

자식 클래스 또는 서브 클래스라고 한다.

부모 클래스 또는 슈퍼 클래스라고 한다.

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) + ")"
```

```
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
```

```
In [58]: myTruck = Truck("Tisla", "Model S", "white", 10000, 2000)
```

```
In [59]: print(myTruck.getDesc())
```

```
차량 =(Tisla,Model S,white,10000)
```

```
In [60]: myTruck.setMake("Tesla")
```

```
In [61]: myTruck.setPayload(2000)
```

```
In [62]: print(myTruck.getDesc())
```

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

■ 부모 클래스의 생성자를 명시적으로 호출

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

- `super()` 함수의 반환 값을 상위클래스의 객체로 간주
- `super()` 함수는 부모 클래스의 객체 역할을 하는 내장 함수


```
class SchoolMember:
    def __init__(self, name, age):
        self.name = name
        self.age = age
        print('공통 멤버: {}'.format(self.name))
    def tell(self):
        print('이름: "{}" 나이: {}'.format(self.name, self.age), end = ' ')
```

```
class Teacher(SchoolMember):  
    def __init__(self, name, age, salary):  
        SchoolMember.__init__(self, name, age)  
        self.salary = salary  
        print('교수 시작: {}'.format(self.name))  
    def tell(self):  
        SchoolMember.tell(self)  
        print('월급: {}'.format(self.salary))
```

```
class Student(SchoolMember):  
    def __init__(self, name, age, grade):  
        SchoolMember.__init__(self, name, age)  
        self.grade = grade  
        print('학생 시작: {}'.format(self.name))  
    def tell(self):  
        SchoolMember.tell(self)  
        print('학점 : {:.2f}'.format(self.grade))
```

```
In [71]: t = []  
t.append(Teacher('길동1', 30, 300))  
t.append(Teacher('길동2', 35, 350))  
t.append(Teacher('길동3', 40, 400))  
s = []  
s.append(Student('길산1', 21, 3.5))  
s.append(Student('길사2', 22, 4.0))  
s.append(Student('길사3', 23, 4.5))  
print()
```

공통 멤버: 길동1
교수 시작: 길동1
공통 멤버: 길동2
교수 시작: 길동2
공통 멤버: 길동3
교수 시작: 길동3
공통 멤버: 길산1
학생 시작: 길산1
공통 멤버: 길사2
학생 시작: 길사2
공통 멤버: 길사3
학생 시작: 길사3

```
In [72]: members = t + s  
         for member in members:  
             member.tell()
```

```
이름: "길동1" 나이: 30 월급: 300  
이름: "길동2" 나이: 35 월급: 350  
이름: "길동3" 나이: 40 월급: 400  
이름: "길산1" 나이: 21 학점 : 3.50  
이름: "길사2" 나이: 22 학점 : 4.00  
이름: "길사3" 나이: 23 학점 : 4.50
```

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

```
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)
```

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

```
In [117]: s = Student('길동', '12345678', '학생')
```

```
In [118]: s.enrollCourse('자료구조')
```

```
In [119]: print(s)
```

타입=학생
이름=길동
주민번호=12345678
수강과목=자료구조

```
In [123]: t = Teacher('길산', '0987654321', '교수')  
t.assignTeaching("Python")  
print(t)
```

타입=교수
이름=길산
주민번호=0987654321
강의과목=Python

`__str__()` : 객체의 문자열 표현으로 돌려준다.

객체를 `print`할때 `str` 메소드를 호출한다.


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

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

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

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

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

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

    def __str__(self):
        return "\n타입="+self.teacherType+ "\n이름="+self.name+ "\n주민번호="+self.number+\
            "\n강의과목="+str(self.courses)
```

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

```
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.

■ 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.

```
In [130]: jeon = Manager('길동', 200, 100)
```

```
In [131]: print(jeon)
```

이름: 길동; 월급: 200; 보너스: 100

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

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

■ 다형성

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

■ 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 상속받아 Car 클래스와 Truck 클래스를 작성해보자.

```
In [165]: cars = [Truck('truck1'), Truck('truck2'), Car('car1')]

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

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

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

    def drive(self):
        return '운전을 합니다.'

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

```
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())
```

- 다중 상속 : 자식 하나가 여러 부모로부터 상속
 - 클래스 이름을 콤마(,)로 구분하여 적어준다.

```
class A:  
    pass  
  
class B:  
    pass  
  
class C:  
    pass  
  
class D(A, B, C):  
    pass
```

- 다이아몬드 상속 : 다중 상속이 만들어 내는 곤란한 상황
 - D는 B와 C 중 누구의 method()를 물려받게 되는 걸까?

```
class A:
    def method(self):
        print("A")

class B(A):
    def method(self):
        print("B")

class C(A):
    def method(self):
        print("C")

class D(B, C):
    pass
```

■ `__call__(self)` 메소드

- 객체를 함수 호출 방식으로 사용하게 만드는 메소드

```
In [124]: class Callable:
          def __call__(self):
              print('I am called.')
```

```
In [125]: obj = Callable()
```

```
In [126]: obj
```

```
Out[126]: <__main__.Callable at 0x21c61c74b38>
```

```
In [127]: obj()
```

```
I am called.
```