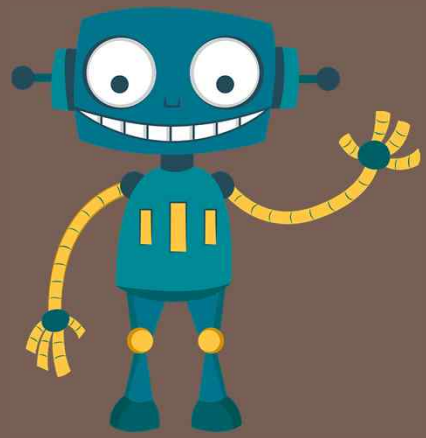
파이썬 익스프레스



12장 상속

Q & A

- 언제라도 질문하세요
  1. 강의시간의 채팅창 (수신인: 모두, 수강생모두에게 답변하기에)
  2. 네이버카페 질의응답게시판

- 강사의 1번 선생님은 여러분들의 질문

앞으로 나는 할 수 있다

1. 나는 부모 클래스를 상속받아서 자식 클래스를 정의할 수 있다
2. 나는 부모 클래스의 메소드를 자식 클래스에서 재정의할 수 있다.
3. 나는 Object 클래스를 이해할 수 있다.
4. 나는 메소드 오버라이딩을 이해하고 사용할 수 있다.
5. 나는 클래스 간의 관계를 파악할 수 있다.

이번 장에서 만들 프로그램

(1) 상속을 이용하여서 각 클래스에 중복된 정보를 부모 클래스로 모아 보자. 구체적인 예로 Car 클래스와 ElectricCar 클래스를 작성해보자.

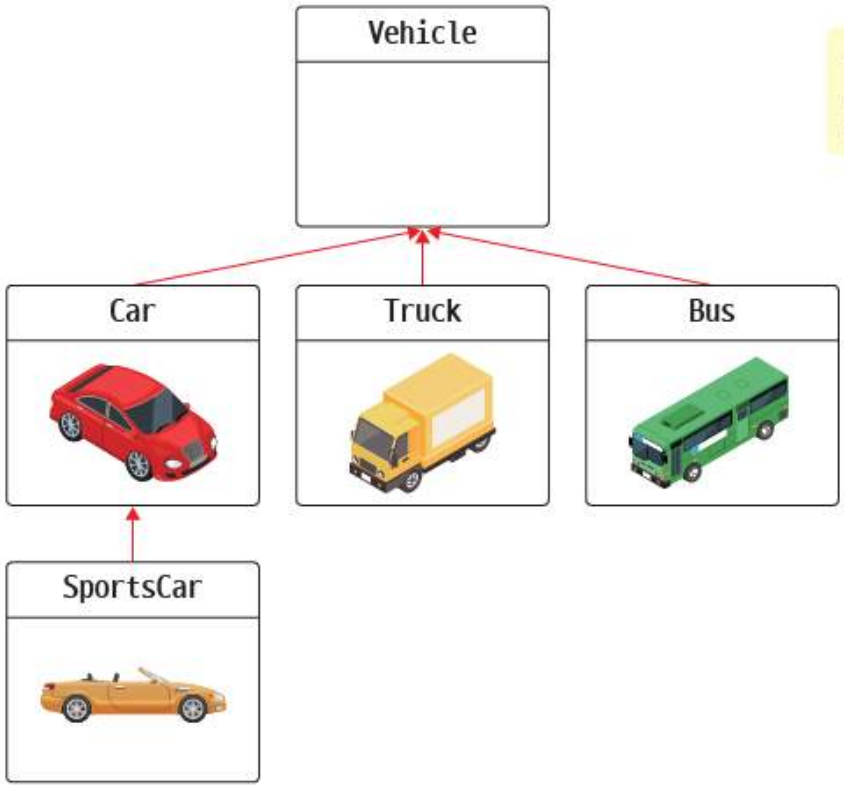
(2) 상속을 사용할 때, 자식 클래스와 부모 클래스의 생성자가 호출되는 순서를 살펴보자.

(3) 부모 클래스의 함수를 오버라이딩(재정의)하여 자식 클래스의 기능을 강력하게 하는 기법을 살펴보자.

상속

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

상속의 예


Vehicle
Car
Truck
Bus
SportsCar


상속에서는 자식 클래스에서
부모 클래스로 화살표를
그립니다.

상속과 is-a 관계

- 객체 지향 프로그래밍에서는 상속이 클래스 간의 "is-a" 관계를 생성하는데 사용된다.

  - 푸들은 강아지이다.
  - 자동차는 차량이다.
  - 꽃은 식물이다.
  - 사각형은 모양이다.

상속의 예

부모 클래스	자식 클래스
Animal(동물)	Lion(사자), Dog(개), Cat(고양이)
Bike(자전거)	MountainBike(산악자전거), RoadBike, TandemBike
Vehicle(탈것)	Car(자동차), Bus(버스), Truck(트럭), Boat(보트), Motocycle(오토바이), Bicycle(자전거)
Student(학생)	GraduateStudent(대학원생), UnderGraduate(학부생)
Person(사람)	Student(학생), Employee(직원)
Shape(도형)	Rectangle(사각형), Triangle(삼각형), Circle(원)

상속 구현하기

Syntax: 상속 정의

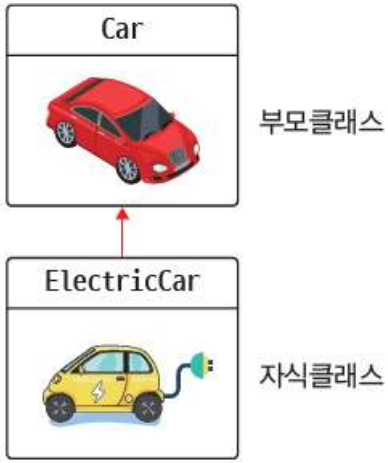
자식 클래스 또는 서브 클래스라고 한다.

Syntax

```
class 자식클래스(부모클래스) :
        def 메소드1(self, ...) :
            ...
        def 메소드2(self, ...) :
            ...
```

부모 클래스 또는 슈퍼 클래스라고 한다.

```
class ElectricCar(Car) :
        def setBatterySize(self, size):
            ...
        def getBatterySize(self):
            ...
```

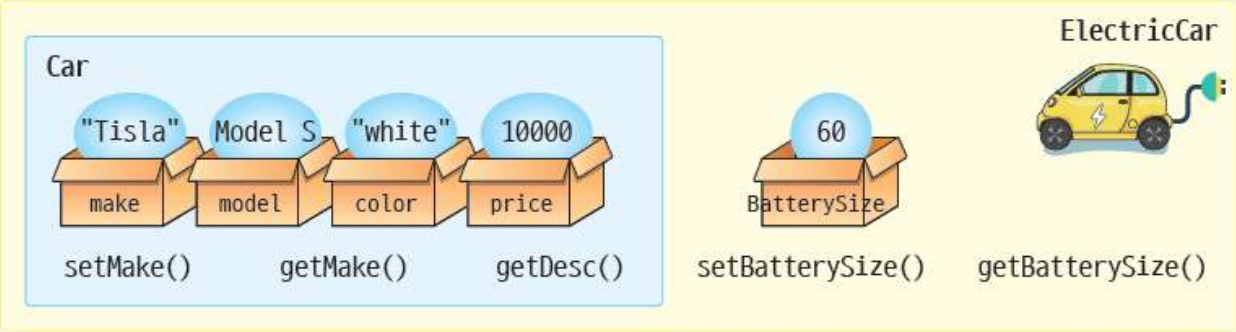


예제 (car.py)

```
# 일반적인 자동차를 나타내는 클래스이다.
class Car:
        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 ElectricCar(Car) :                                    # ①
        def __init__(self, make, model, color, price, batterySize):
                super().__init__(make, model, color, price)    # ②
                self.batterySize=batterySize                                    # ③

        def setBatterySize(self,  batterySize):          # 설정자 메소드
                self.batterySize=batterySize

        def getBtterySize(self):                                  # 접근자 메소드
                return self.batterySize
```

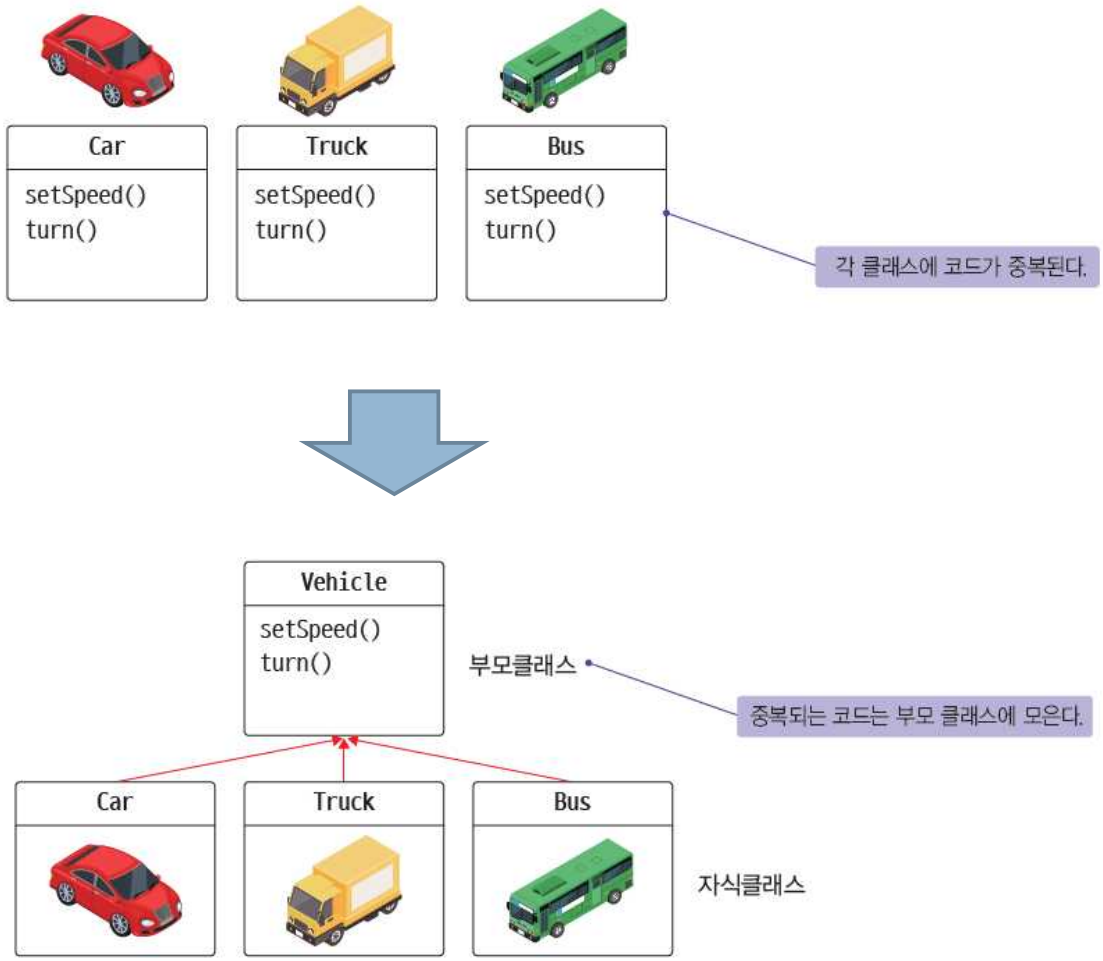


예제

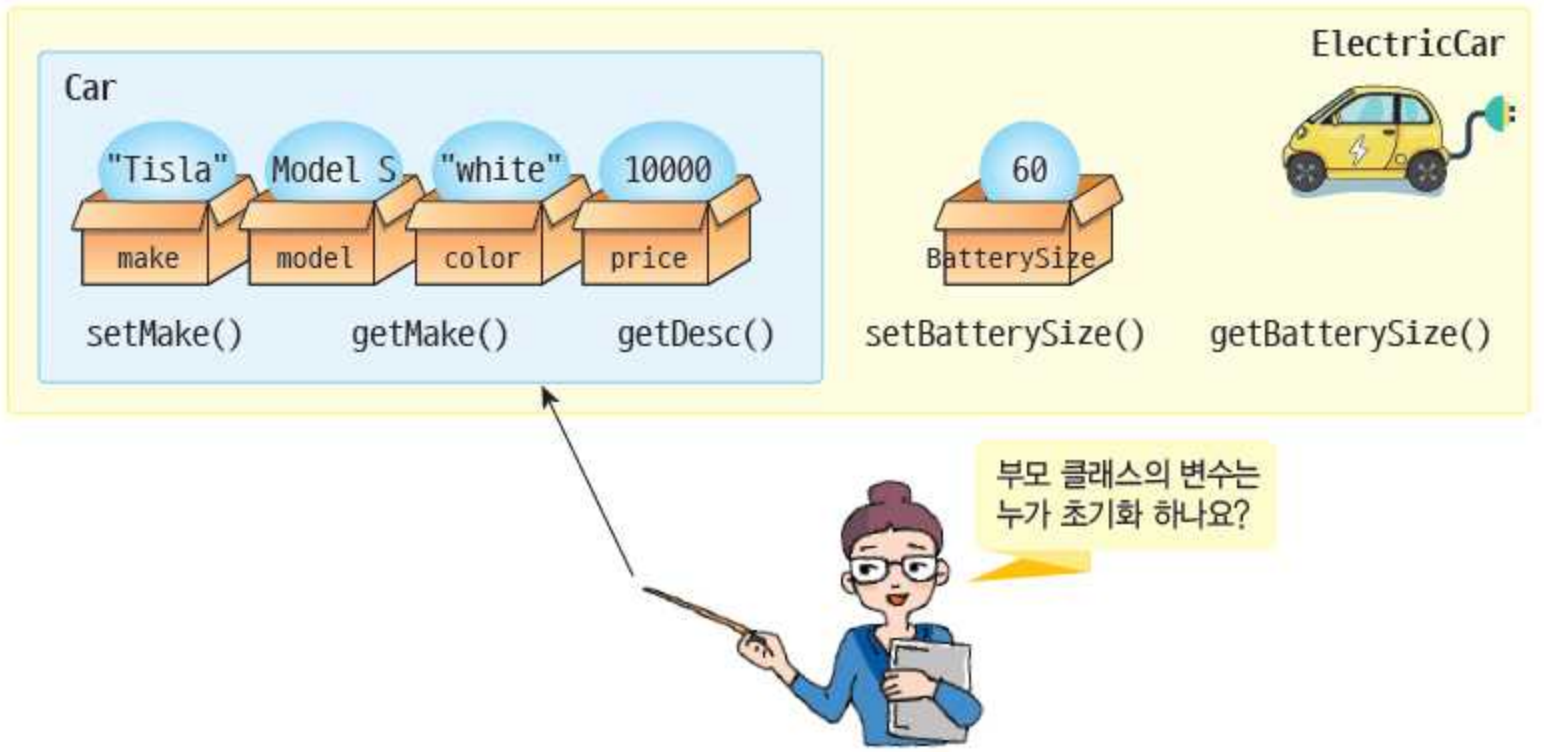
```
def main():                                                    # main() 함수 정의

        myCar = ElectricCar("Tisla", "Model S", "white", 10000, 0)        #
        myCar.setMake("Tesla")                                 # 설정자 메소드 호출
        myCar.setBatterSize(60)                                # 설정자 메소드 호출
        print(myCar.getDesc())      # 전기차 객체를 문자열로 출력

main()
```

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

왜 상속을 사용하는가?


Car
setSpeed()
turn()
Truck
setSpeed()
turn()
Bus
setSpeed()
turn()
각 클래스에 코드가 중복된다.
Vehicle
setSpeed()
turn()
부모클래스
중복되는 코드는 부모 클래스에 모은다.
Car
Truck
Bus
자식클래스

부모 클래스의 생성자 호출



생성자를 호출하지 않으면 오류

```
class ElectricCar(Car) :
        def __init__(self, make, model, color, price, batterySize):
                super().__init__(make, model, color, price)
                self.batterySize=batterySize
```

생성자를 호출하지 않으면 오류 (animal.py)

```
class Animal:
    def __init__(self, age=0):
        self.age=age

    def eat(self):
        print("동물이 먹고 있습니다. ")


# 부모 클래스의 생성자를 호출하지 않았다!
class Dog(Animal):
    def __init__(self, age=0, name=""):
        self.name=name

# 부모 클래스의 생성자가 호출되지 않아서 age 변수가 생성되지 않았다.

d = Dog();
print(d.age)
```

type()과 isinstance() 함수 (animal_change.py)

```
...
x = Animal();
y = Dog();
print(type(x))
print(type(y))
```

```
<class '__main__.Animal'>
<class '__main__.Dog'>
```

학생> type() 설명은 어디있나요?

교수>

1. Console창에서 help(type)
2. https://docs.python.org 에서 type 검색

type()과 isinstance() 함수

```
...
x = Animal()
y = Dog()
print(isinstance(x, Animal), isinstance(y, Animal))
```

```
True True
```

In : print(isinstance(x, Dog), isinstance(y, Dog))

private 멤버 부모 클래스 (private_change.py)

```
class Parent(object):
    def __init__(self):
        self.__money = 100

class Child(Parent):
    def __init__(self):
private가 되어서 자식 클래스에서 사용할 수 없다.

        super().__init__()

obj = Child()
print(obj.__money) # 오류
```

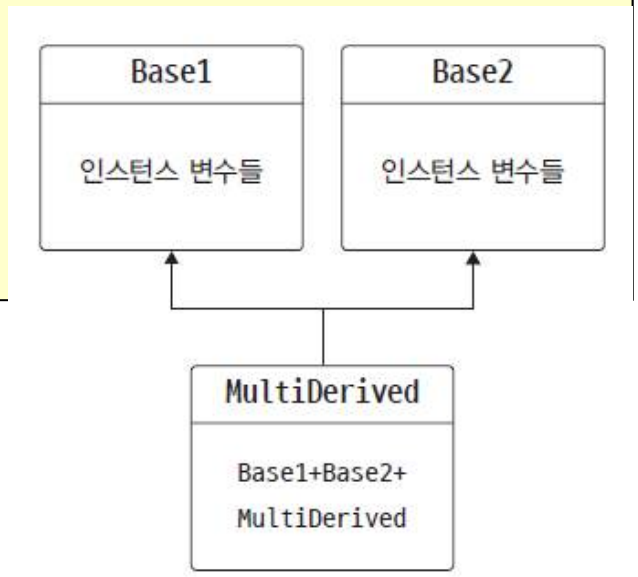
```
...
AttributeError: 'Child' object has no attribute 'money'
```

다중 상속

다중 상속을 C++ 처럼 함

Java는 class와 interface를 사용하여 다중 상속 효과 제공

```
class Base1:
    pass

class Base2:
    pass

class MultiDerived(Base1, Base2):
    pass
```



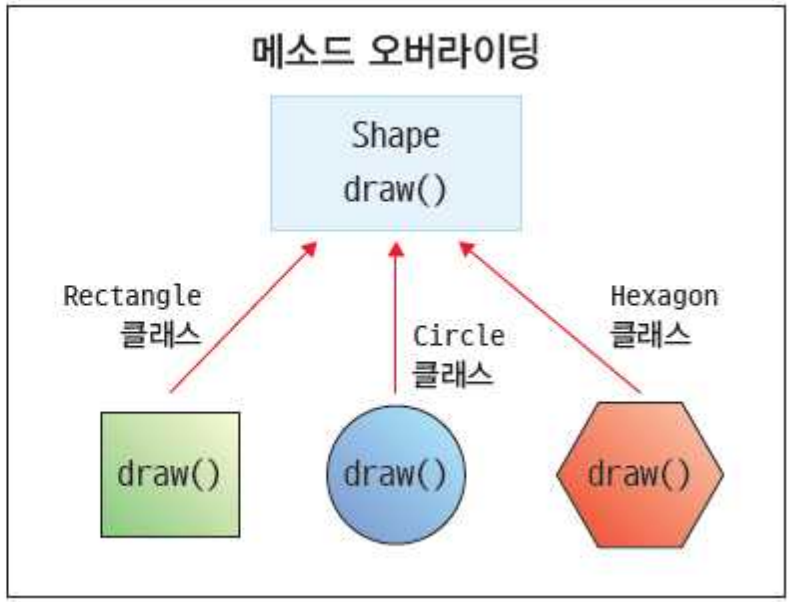
다중 상속의 예 (student_change.py)

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

    def show(self):
        print(self.name, self.age)

 class Student:
    def __init__(self, id):
        self.id = id

    def getId(self):
        return self.id

class CollegeStudent(Person, Student):
    def __init__(self, name, age, id):
        Person.__init__(self, name, age)
        Student.__init__(self, id)

obj = CollegeStudent('Kim', 22, '100036')
obj.show()
print(obj.getId())
dir(obj)
```

```
Kim 22
100036
```

메소드 오버라이딩

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



shape.py

```
import math

class Shape:
    def __init__(self):
        pass

    def draw(self):
        print("draw()가 호출됨")

    def get_area(self):
        print("get_area()가 호출됨")

class Circle(Shape):
    def __init__(self, radius=0):
        super().__init__()
        self.radius = radius

    def draw(self):
        print("원을 그립니다.")

    def get_area(self):
        return math.pi * self.radius ** 2

c = Circle(10)
c.draw()
print("원의 면적:", c.get_area())
```

```
원을 그립니다.
원의 면적: 314.1592653589793
```

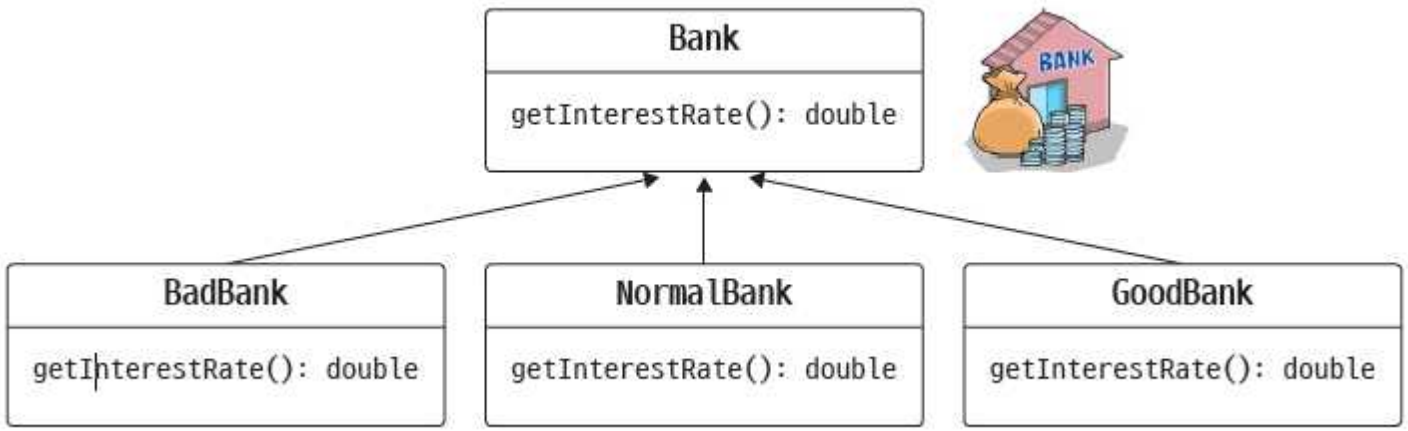
예제 (메소드 오버라이딩인 경우 부모 메소드 호출방법)

```
class Circle(Shape):
                ...
    def draw(self):
        super().draw()              # 부모 클래스의 draw()가 호출된다.
        print("원을 그립니다.")
```

```
draw()가 호출됨
원을 그립니다.
```

Lab: Bank 클래스 (bank.py)

- 은행에서 대출을 받을 때, 은행마다 대출 이자가 다르다. 이것을 메소드 오버라이딩으로 깔금하게 해결하여 보자.




```
BadBank의 이자율: 10.0
NormalBank의 이자율: 5.0
GoodBank의 이자율: 3.0
```

```
class Bank():
        def getInterestRate(self):
                return 0.0

class BadBank(Bank):
        def getInterestRate(self):
                return 10.0;

class NormalBank(Bank):
        def getInterestRate(self):
                return 5.0;

class GoodBank(Bank):
        def getInterestRate(self):
                return 3.0;

b1 = BadBank()
b2 = NormalBank()
b3 = GoodBank()
print("BadBank의 이자율: " + str(b1.getInterestRate()))
print("NormalBank의 이자율: " + str(b2.getInterestRate()))
print("GoodBank의 이자율: " + str(b3.getInterestRate()))
```

Lab: 직원과 매니저 (employee.py)

- 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
```

Solution

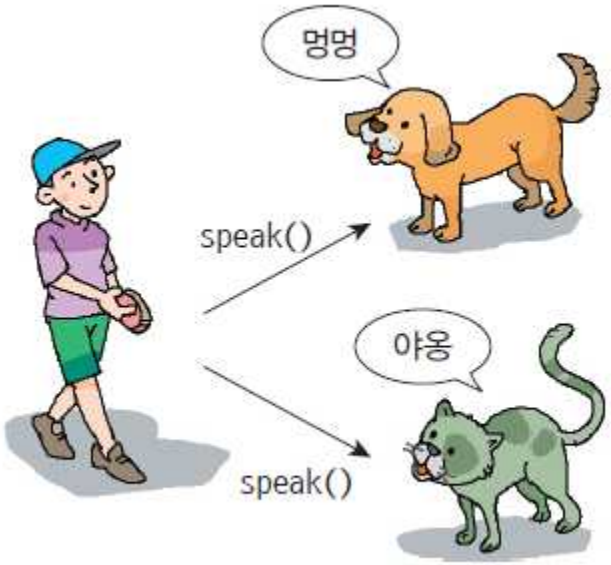
학생> __str__()은 무엇인가요?

교수> slide 35와 37

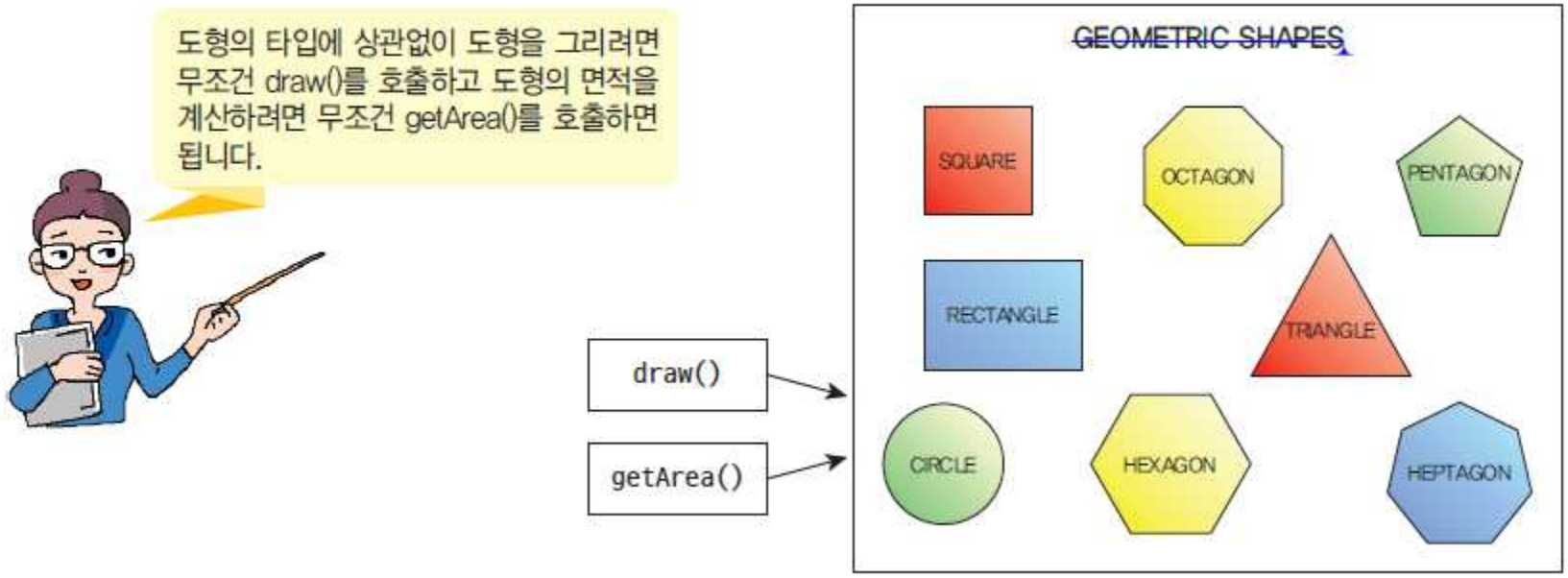
```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

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

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

다형성

- **다형성(polymorphism)**은 **"많은(poly)+모양(morph)"**이라는 의미로서 주로 프로그래밍 언어에서 하나의 식별자로 다양한 타입(클래스)을 처리하는 것을 의미한다.
- 객체지향 프로그래밍의 4가지 기능 중 4번째
  캡슐화, 정보은닉, 상속, 다형성
- 상속기능 활용 사례



다형성의 예


도형의 타입에 상관없이 도형을 그리려면 무조건 draw()를 호출하고 도형의 면적을 계산하려면 무조건 getArea()를 호출하면 됩니다.
draw()
getArea()
GEOMETRIC SHAPES
SQUARE
OCTAGON
PENTAGON
RECTANGLE
TRIANGLE
CIRCLE
HEXAGON
HEPTAGON

상속과 다형성 (shape2.py)

```
class Shape:
    def __init__(self, name):
        self.name = name
    def getArea(self):
        raise NotImplementedError("이것은 추상메소드입니다. ")

class Circle(Shape):
    def __init__(self, name, radius):
        super().__init__(name)
        self.radius = radius

    def getArea(self):
        return  3.141592*self.radius**2

class Rectangle(Shape):
    def __init__(self, name, width, height):
        super().__init__(name)
        self.width = width
        self.height = height

    def getArea(self):
        return  self.width*self.height
```

상속과 다형성

학생> 교수님~ 이 코드가 왜 다형성인가요?

교수> shape2_change.py의 getArea(s) 에서 매개변수 s의 자료형이 C++이나 Java에서는 부모class로 다양한 자식 class를 받는 경우를 다형성이라고 합니다.

C++과 Java는 자료형이 static type이지만 파이썬은 자료형이 dynamic type이라서 부모class 자료형 선언 없이 다형성기능을 제공합니다

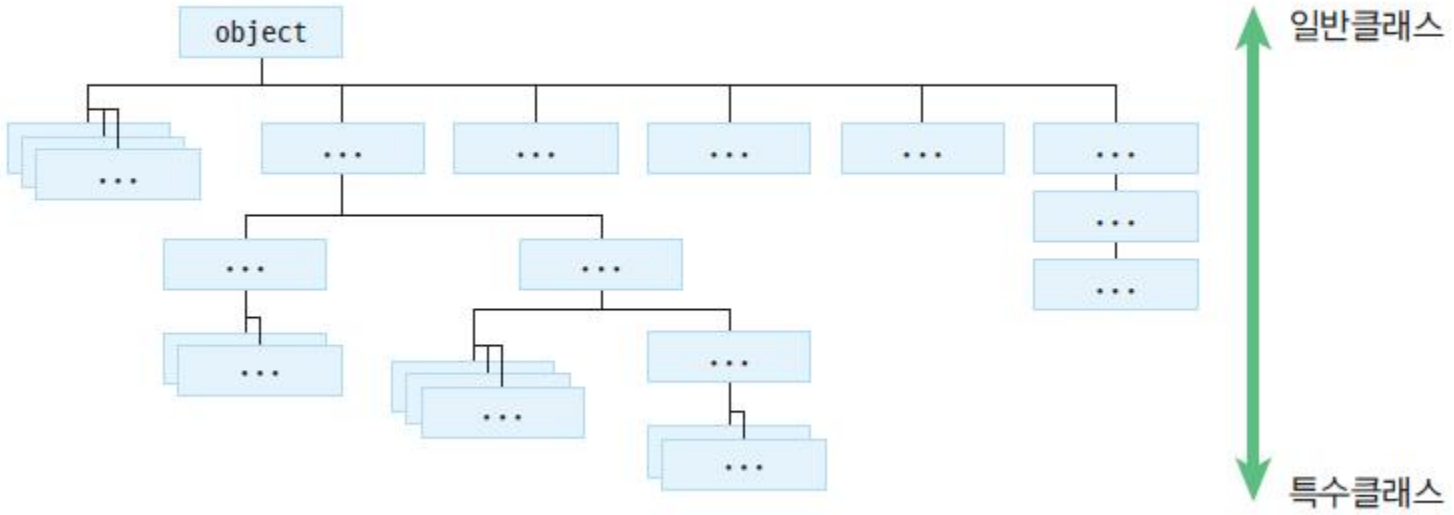
```
shapeList = [ Circle("c1", 10), Rectangle("r1", 10, 10) ]
for s in shapeList:
        print(s.getArea())
```

```
314.1592
100
```

object 클래스

- 모든 클래스의 맨 위에는 object 클래스가 있다고 생각하면 된다.

=> 따라서 모든 클래스는 object 클래스의 변수와 메소드를 상속받음



object 클래스의 메소드

In: help (object)

In : obj = object()

In : dir(obj)

주의 : 통상 Class 이름은 대문자로 시작하는데 object클래스는 소문자로 시작

객체(object)이름은 소문자로 시작

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

__repr__() 메소드 (book.py)

```
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
```

repr(book) 도 print(book)과 동일한 효과

__str__() 메소드 (mytime.py)

```
class MyTime:
        def __init__(self, hour, minute, second=0):
                self.hour = hour
                self.minute = minute
                self.second = second
        def __str__(self):
                return '%.2d:%.2d:%.2d' % (self.hour, self.minute, self.second)

time = MyTime(10, 25)
print(time)
```

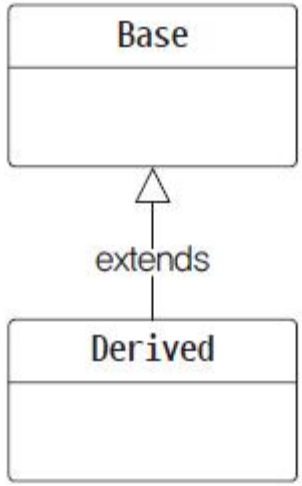
```
10:25:00
```

str(time) 도 print(time)과 동일한 효과

__str__()와 __repr__()차이점은 교재 569페이지에 자세한 설명

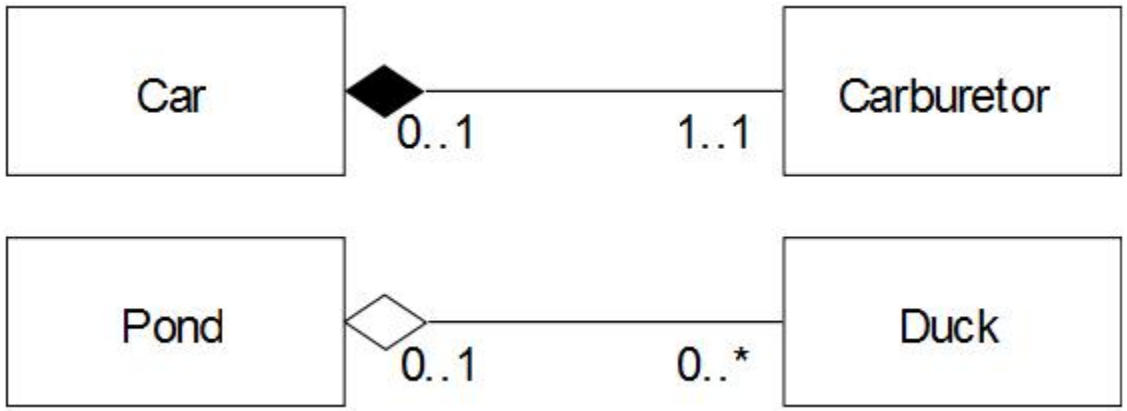
클래스 관계: 상속

- is-a 관계: 상속
  - 승용차는 차량의 일종이다(Car is a Vehicle).
  - 강아지는 동물의 일종이다(Dog is an animal).
  - 원은 도형의 일종이다(Circle is a shape)



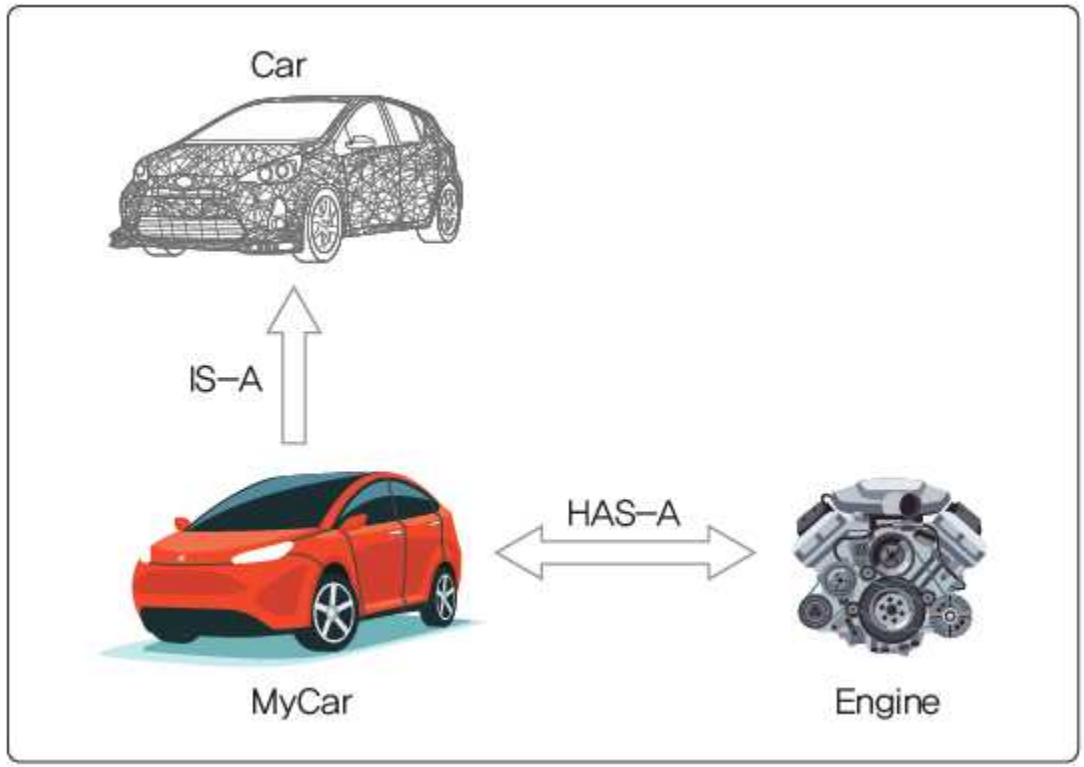
클래스 관계:구성

- has-a 관계: 구성
  - 도서관은 책을 가지고 있다(Library has a book).
  - 거실은 소파를 가지고 있다(Living room has a sofa).



Is-a관계 vs has-a관계 vs 관계없음

객체간의 관계는 3가지만 있음



예 (animal2.py)

- Dog는 Animal의 is-a관계
- Dog는 Person의 has-a관계
- 부모클래스 object는 생략가능 class Animal(object): => class Animal : animal2_change.py

```
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 (card.py)

- 카드를 나타내는 Card 클래스를 작성하고 52개의 Card 객체를 가지고 있는 Deck 클래스를 작성한다. 각 클래스의 __str__() 메소드를 구현하여서 덱 안에 들어 있는 카드를 다음과 같이 출력한다.
- https://en.wikipedia.org/wiki/Standard_52-card_deck
  deck = pack of cards
- Has-a 관계

```
['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.
```

Lab: 학생과 강사 (person.py)

- 일반적인 사람을 나타내는 Person 클래스를 정의한다. Person 클래스를 상속받아서 학생을 나타내는 클래스 Student와 선생님을 나타내는 클래스 Teacher를 정의한다.
- Person과 Student는 is-a관계
- Person과 Teacher는 is-a관계
- Student와 Teacher는 무관계

```
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)
```

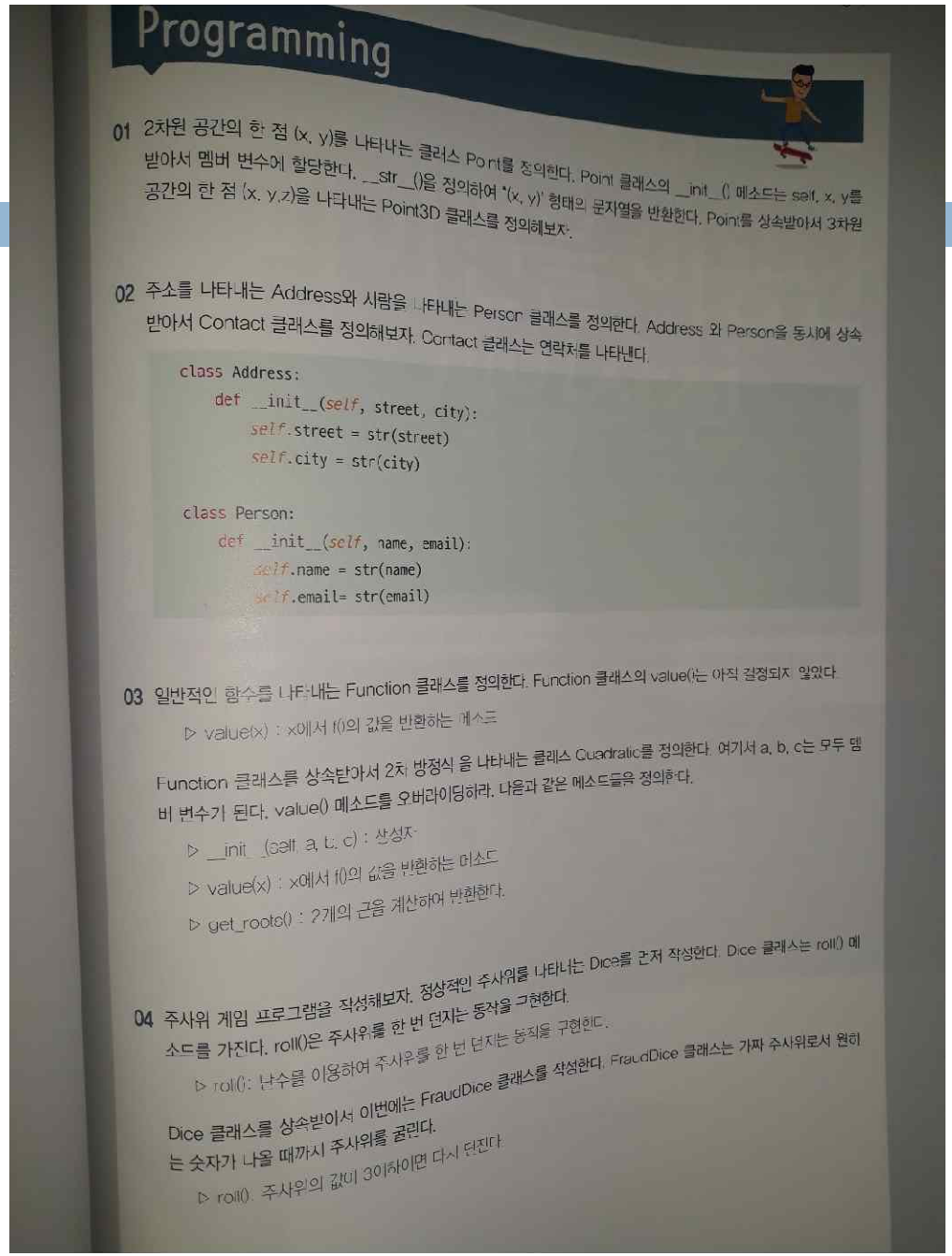
이제 나는 할 수 있다

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

HW12장-1,2

- HW12장-1,2 게시판에 업로드

프로그래밍문제_1



Programming

01 2차원 공간의 한 점 (x, y)를 나타내는 클래스 Point를 정의한다. Point 클래스의 __init__() 메소드는 self, x, y를 받아서 멤버 변수에 할당한다. __str__()을 정의하여 '(x, y)' 형태의 문자열을 반환한다. Point를 상속받아서 3차원 공간의 한 점 (x, y,z)을 나타내는 Point3D 클래스를 정의해보자.

02 주소를 나타내는 Address와 사람을 나타내는 Person 클래스를 정의한다. Address 와 Person을 동시에 상속받아서 Contact 클래스를 정의해보자. Contact 클래스는 연락처를 나타낸다.

```
class Address:
    def __init__(self, street, city):
        self.street = str(street)
        self.city = str(city)

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

03 일반적인 함수를 나타내는 Function 클래스를 정의한다. Function 클래스의 value()는 아직 결정되지 않았다.

▷ value(x) : x에서 f()의 값을 반환하는 메소드

Function 클래스를 상속받아서 2차 방정식 을 나타내는 클래스 Quadratic를 정의한다. 여기서 a, b, c는 모두 멤버 변수가 된다. value() 메소드를 오버라이딩하라. 다음과 같은 메소드들을 정의한다.

▷ __init__(self, a, b, c) : 생성자

▷ value(x) : x에서 f()의 값을 반환하는 메소드

▷ get_roots() : 2개의 근을 계산하여 반환한다.

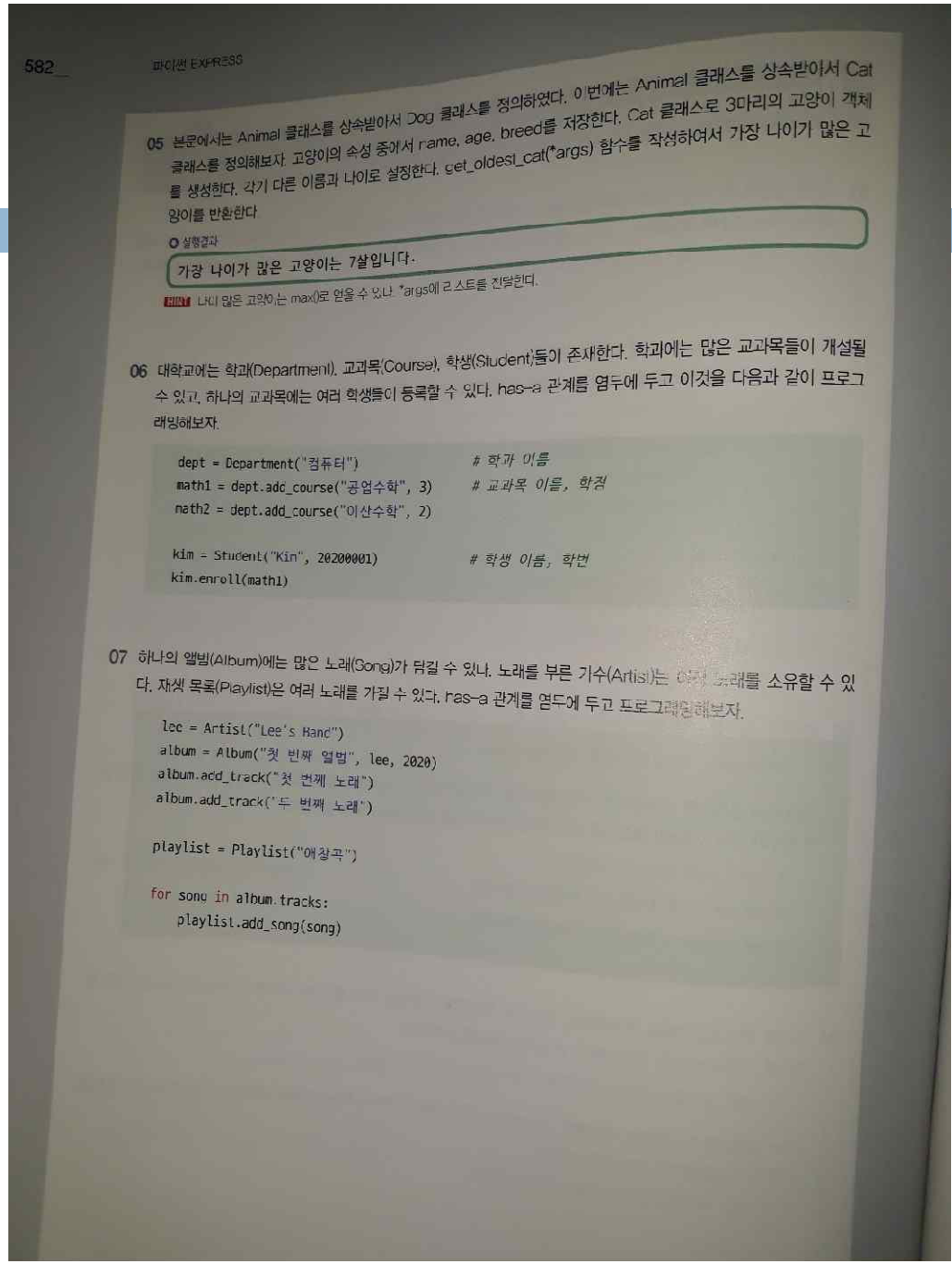
04 주사위 게임 프로그램을 작성해보자. 정상적인 주사위를 나타내는 Dice를 먼저 작성한다. Dice 클래스는 roll() 메소드를 가진다. roll()은 주사위를 한 번 던지는 동작을 구현한다.

▷ roll(): 난수를 이용하여 주사위를 한 번 던지는 동작을 구현한다.

Dice 클래스를 상속받아서 이번에는 FraudDice 클래스를 작성한다. FraudDice 클래스는 가짜 주사위로서 원하는 숫자가 나올 때까지 주사위를 굴린다.

▷ roll(). 주사위의 값이 3이하이면 다시 던진다.

프로그래밍문제-2



582_ 파이썬 EXPRESS

05 본문에서는 Animal 클래스를 상속받아서 Dog 클래스를 정의하였다. 이번에는 Animal 클래스를 상속받아서 Cat 클래스를 정의해보자. 고양이의 속성 중에서 name, age, breed를 저장한다. Cat 클래스로 3마리의 고양이 객체를 생성한다. 각기 다른 이름과 나이로 설정한다. get_oldest_cat(*args) 함수를 작성하여서 가장 나이가 많은 고양이를 반환한다

○ 실행결과

```
가장 나이가 많은 고양이는 7살입니다.
```

HINT 나이 많은 고양이는 max()로 얻을 수 있다. *args에 리스트를 전달한다.

06 대학교에는 학과(Department), 교과목(Course), 학생(Student)들이 존재한다. 학과에는 많은 교과목들이 개설될 수 있고, 하나의 교과목에는 여러 학생들이 등록할 수 있다. has-a 관계를 염두에 두고 이것을 다음과 같이 프로그래밍해보자.

```
dept = Department("컴퓨터")                # 학과 이름
math1 = dept.add_course("공업수학", 3)     # 교과목 이름, 학점
math2 = dept.add_course("이산수학", 2)

kim = Student("Kim", 20200001)             # 학생 이름, 학번
kim.enroll(math1)
```

07 하나의 앨범(Album)에는 많은 노래(Song)가 담길 수 있나. 노래를 부른 가수(Artist)는 여러 노래를 소유할 수 있다. 재생 목록(Playlist)은 여러 노래를 가질 수 있다. has-a 관계를 염두에 두고 프로그래밍해보자.

```
lee = Artist("Lee's Band")
album = Album("첫 번째 앨범", lee, 2020)
album.add_track("첫 번째 노래")
album.add_track("두 번째 노래")

playlist = Playlist("애창곡")

for song in album.tracks:
    playlist.add_song(song)
```