파이썬 Express

12장 상속

학습 목표

- 부모 클래스를 상속받아서 자식 클래스를 정의할 수 있다
- 부모 클래스의 메소드를 자식 클래스에서 재정의할 수 있다.
- Object 클래스를 이해할 수 있다.
- 메소드 오버라이딩을 이해하고 사용할 수 있다.
- 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.

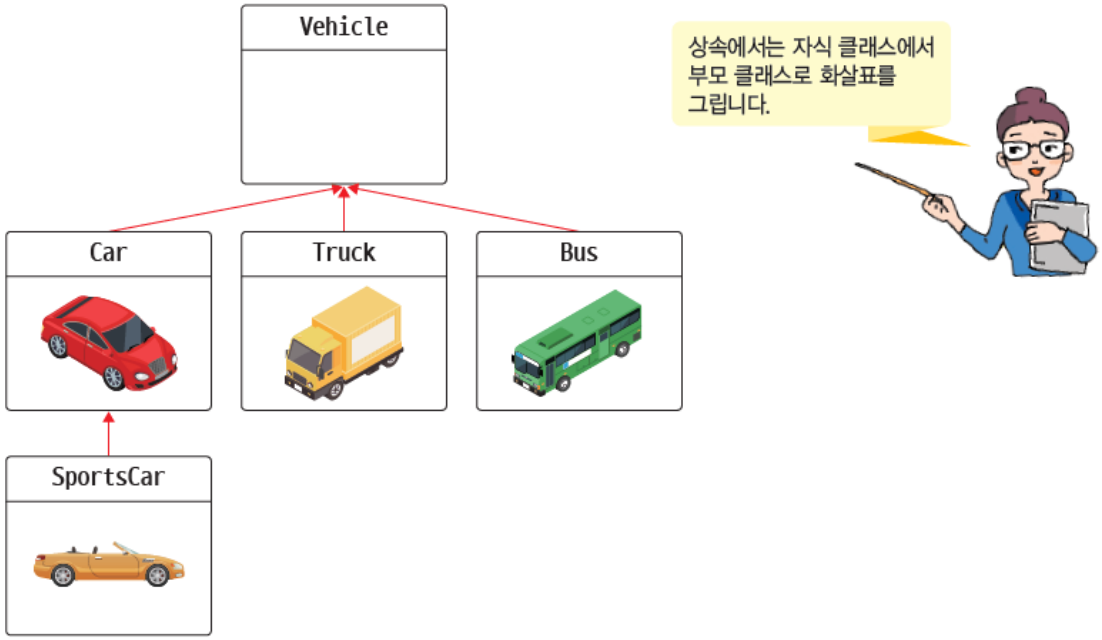
(3) 부모 클래스의 함수를 오버라이딩(재정의)하여 자식 클래스의 기능을 강력하게 하는 기법을 살펴보자.

상속의 개념

- 상속(inheritance) : 기존에 존재하는 클래스로 부터 코드와 데이터를 이어받고 자신이 필요한 기능을 추가하는 기법.
- 이미 작성된 검증된 소프트웨어를 재사용할 수 있어서 신뢰성 있는 소프트웨어를 손쉽게 개발, 유지보수 할 수 있게 해주는 중요한 기술

상속의 예

- 유사한 객체들은 공통된 속성과 동작을 공유한다. 자동차, 버스, 트럭은 모두 바퀴가 있고 연료를 사용하며 사람들을 수송할 수 있다. 따라서 이들은 동일한 특성(변수)과 동작(함수)을 많이 가질 것이다. 일반적인 차량을 나타내는 클래스 Vehicle을 작성하고 이 Vehicle에 공통적인 코드를 놓고, Vehicle 클래스를 상속받아서 Car, Bus, Truck 클래스를 작성하면 한결 쉬울 것이다.



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

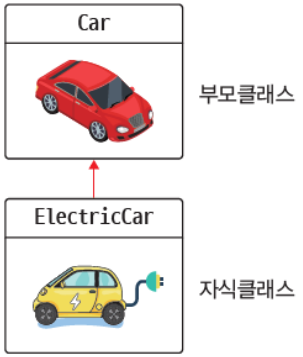
자식 클래스 또는 서브 클래스라고 한다.

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):
        ...
```



상속 구현하기

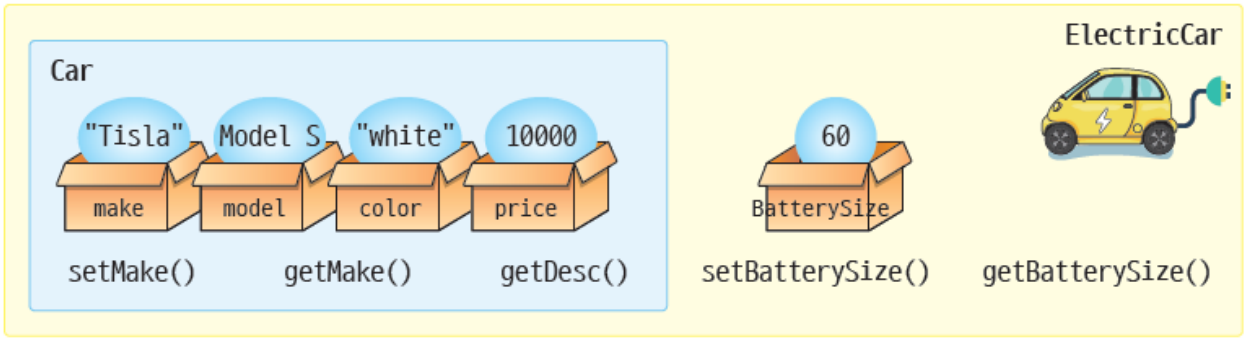
- 예. Car 클래스로 부터 상속을 받아서 전기차를 나타내는 ElectricCar 클래스를 정의

p551.py

```
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


        …
        …
```

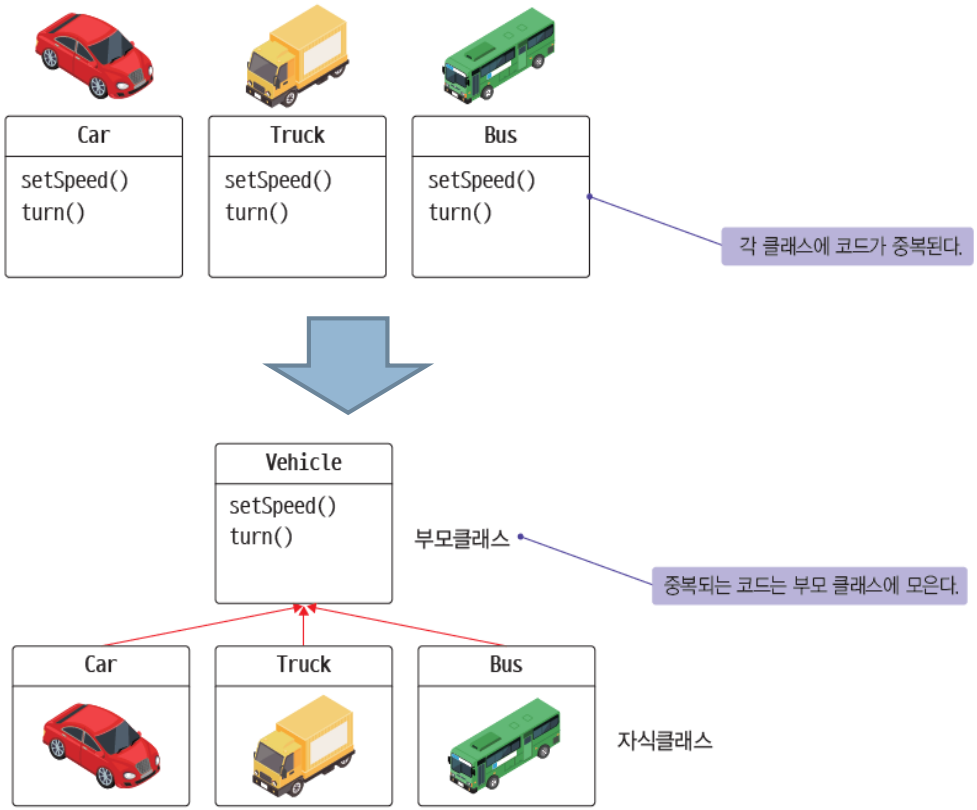
무엇이 상속되는가

- 부모 클래스의 인스턴스 변수와 메소드가 자식 클래스로 상속된다. 자식 클래스는 부모 클래스의 인스턴스 변수와 메소드를 자유롭게 사용할 수 있다.
- 자식 클래스는 필요하면 자신만의 인스턴스 변수와 메소드를 추가시킬 수도 있고 부모 클래스에 이미 존재하는 메소드를 새롭게 정의하여 사용할 수도 있다.

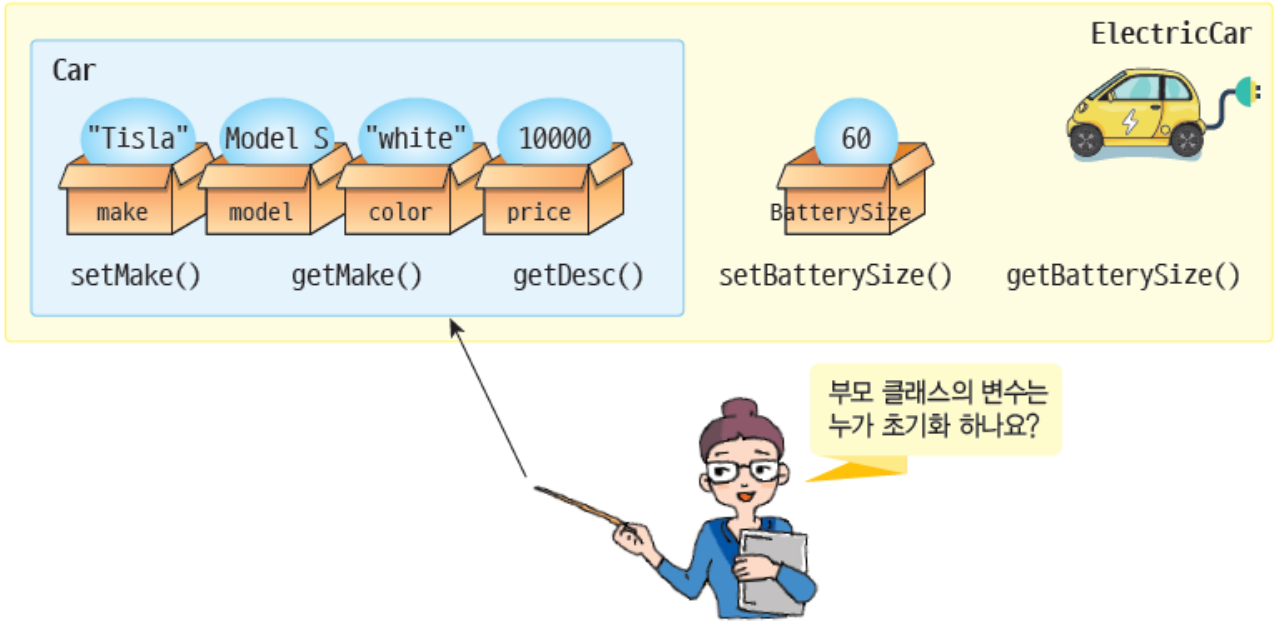


왜 상속을 사용하는가

- 이미 존재하는 클래스의 인스턴스 변수와 메소드를 재사용할 수 있다.
- 중복되는 코드를 줄일 수 있다.



부모 클래스의 생성자 호출



- super().__init__()와 같이 명시적으로 호출한다.

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

생성자를 호출하지 않으면 오류

- 자식 클래스가 부모 클래스의 생성자를 명시적으로 호출하지 않으면 변수 age는 생성되지 않는다.

p556.py

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

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

class Dog(Animal):
    def __init__(self, age=0, name=""):
        self.name=name

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

부모 클래스의 생성자를 호출하지 않았다!

부모 클래스의 생성자가 호출되지 않아서 age 변수가 생성되지 않았다.

type()과 isinstance() 함수

- type() : 현재 객체를 생성한 클래스를 출력
- isinstance() : 객체를 만든 클래스이면 True를 반환

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

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

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

```
True True
```

부모 클래스의 private 멤버 부모 클래스

- 부모의 인스턴스 변수가 자식 클래스에 상속되기를 원치 않을 때.
- private 변수(__ 추가)로 지정

p557.py

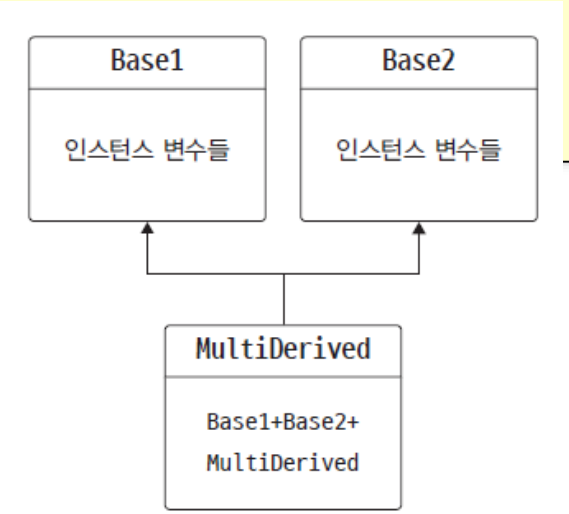
```
class Parent(object):
    def __init__(self):
        self.__money = 100

class Child(Parent):
    def __init__(self):
        super().__init__()

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

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

다중 상속

- 자식 클래스가 둘 이상의 부모 클래스로 부터 상속

```
class Base1:
    pass

class Base2:
    pass

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



다중 상속

p558.py

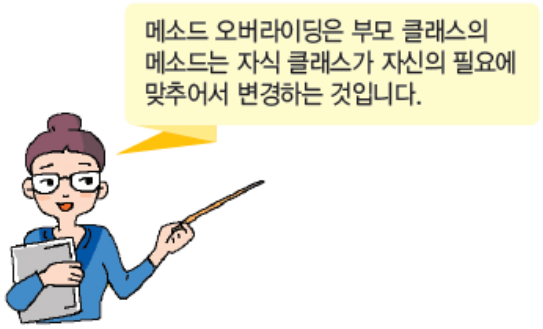
```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    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)
…
```

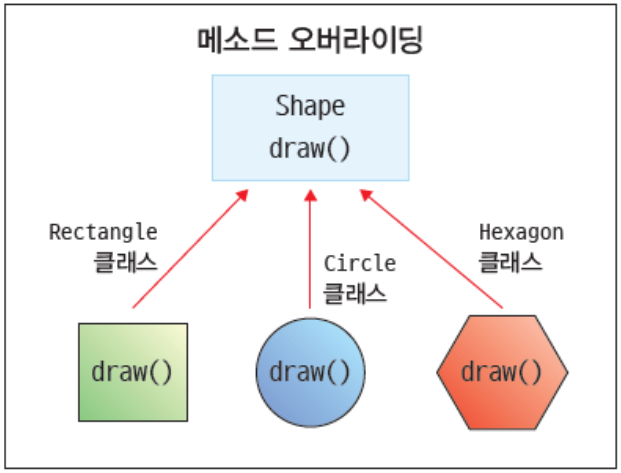
부모 클래스가 2개 이상 일때에는 super()를 사용하지 말고 클래스 이름으로 생성자를 호출

```
Kim 22
100036
```

메소드 오버라이딩

- 자식 클래스가 부모 클래스의 메소드 중에서 필요한 것을 재정의(override)하는 것
- 예. Shape 클래스의 draw() 메소드는 어느 자식 클래스에서 상속을 받는냐에 따라 기능이 달라져야 한다. 즉 재정의가 필요





메소드 오버라이딩

p560.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):                 #오버라이딩(overriding)
        print("원을 그립니다.")
    def get_area(self):             #오버라이딩(overriding)
        return math.pi * self.radius ** 2
…
```

부모 클래스 메소드 호출하기

- super()를 호출하여 부모 클래스를 얻은 후에 메소드를 호출한다.

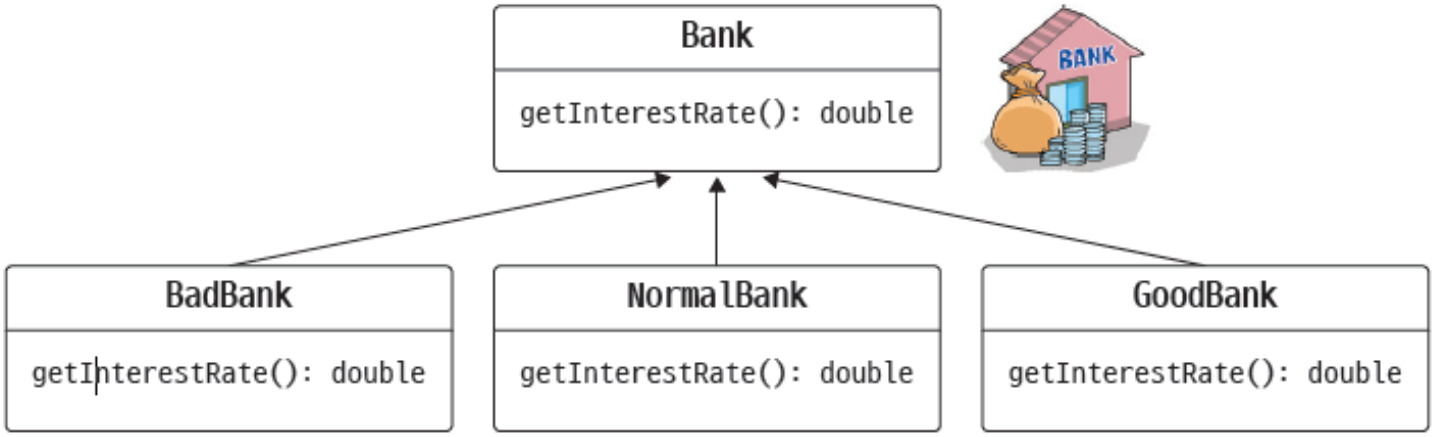
p561.py

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

```
draw()가 호출됨
원을 그립니다.
```

Lab: 은행의 이율 계산하기

- 은행에서 대출을 받을 때, 은행마다 대출 이자가 다르다. Bank 클래스에는 getInterestRate()라는 메소드를 두어서 이자율을 반환한다. Badbank, NoramlBank, GoodBank 클래스에서는 getInterestRate()를 오버라이드하여 이자율을 각각 10%, 5%, 3%로 반환하도록 한다.



Lab: 은행의 이율 계산하기

p562.py

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

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

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

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

b1 = BadBank()
…
```

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

Lab: 직원과 매니저

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

  name – 이름(문자열)

  salary – 월급(정수형)

  getSalary() – 월급을 반환하는 메서드
- Manager 클래스

  name – 이름(문자열)

  salary – 월급(정수형)

  bonus – 보너스(정수형)

  getSalary() – 월급과 보너스를 합하여 반환하는 메서드

Lab: 직원과 매니저

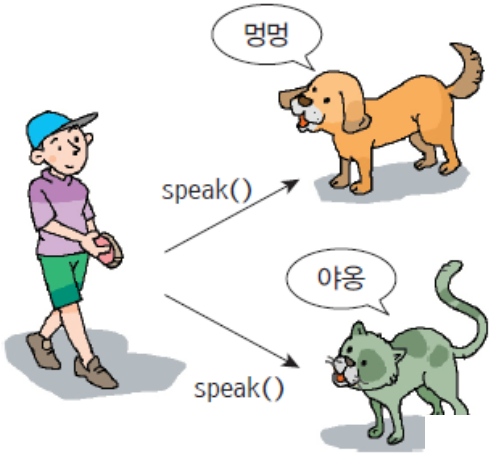
p563.py

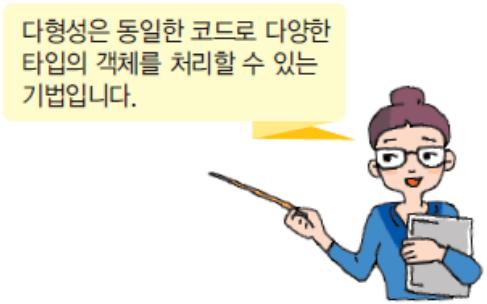
```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
    def  getSalary(self):
        return salary

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)
print(f"총 급여 : {kim.getSalary()}원")
```

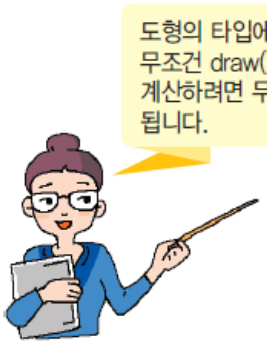
```
이름: 김철수; 월급: 2000000; 보너스: 1000000
총 급여 : 3000000원
```

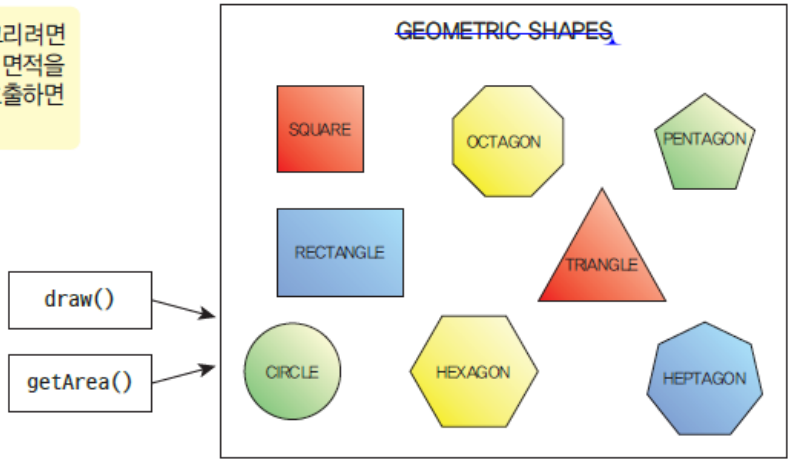
다형성과 동적 바인딩

- 다형성(polymorphism) : “많은(poly)+모양(morph)“. 하나의 식별자로 다양한 타입(클래스)을 처리하는 것









상속과 다형성

- 동일한 메서드를 호출하여도 자식 클래스에 따라 다르게 처리할 수 있다.

p565.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):
…
```

```
314.1592
100
```

파이썬의 내장 함수들

- 파이썬은 다형성을 최대로 이용하고 있다. 예를 들어서 내장 함수 len()은 타입을 가리지 않고 동작한다. len()은 리스트나 문자열, 딕셔너리 객체에서 동작된다. 이것도 일종의 다형성이다.

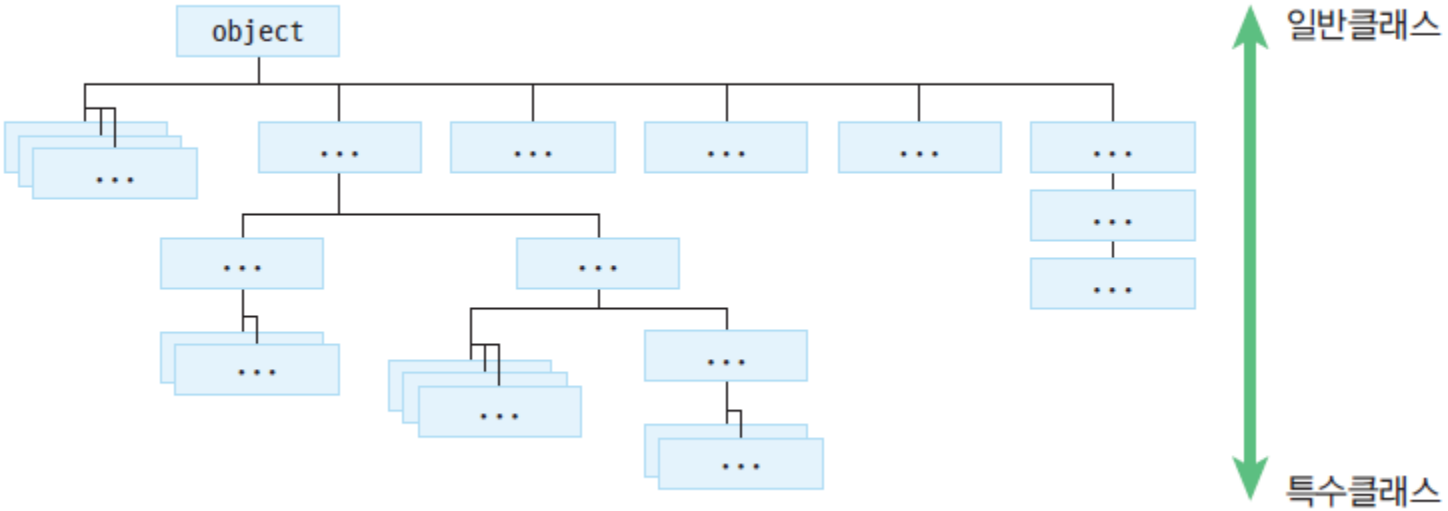
p566.py

```
mylist = [1, 2, 3]                # 리스트
print("리스트의 길이=", len(mylist))

s = "This is a sentense"          # 문자열
print("문자열의 길이=", len(s))

d = {'aaa': 1, 'bbb': 2}          # 딕셔너리
print("딕셔너리의 길이=", len(d))
```

```
리스트의 길이= 3
문자열의 길이= 18
딕셔너리의 길이= 2
```

object 클래스

- 클래스를 작성할 때 부모 클래스를 명시적으로 지정하지 않으면 object 클래스의 자식 클래스로 암묵적으로 간주된다.
- 모든 클래스의 가장 위에는 object 클래스가 있다고 생각하면 된다.



object 클래스

- object 클래스는 모든 메서드에 공통적인 메소드를 가지고 있다.

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

- 객체가 가진 정보를 한 줄의 문자열로 만들어서 반환한다.
- 디버깅에 유용

p568.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
```

__str__() 메소드

- 객체의 문자열 표현을 반환한다

p569_str.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__()와 __repr__ 차이점

- __repr__()은 객체 정보들을 확실하게 확인하기 위한 출력
- __str__()은 객체를 가독성있게 텍스트 형태로 출력하는 것

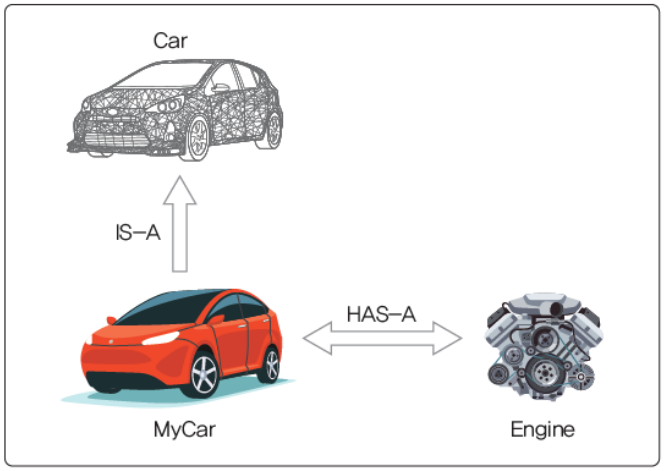
p569_xxx.py

```
import datetime
today = datetime.datetime.now()
print(str(today))
print(repr(today))
```

```
2021-07-08 17:21:16.110103
datetime.datetime(2021, 7, 8, 17, 21, 16, 110103)
```

상속과 구성

- is-a : 상속 관계
  - 승용차는 차량의 일종이다(Car is a Vehicle).
  - 강아지는 동물의 일종이다(Dog is an animal).
  - 원은 도형의 일종이다(Circle is a shape)
- has-a : 포함 관계
  - 도서관은 책을 가지고 있다(Library has a book).
  - 거실은 소파를 가지고 있다(Living room has a sofa).



자동차는 Car 클래스로 부터 상속받을 수 있고(is-a) 내부에 Engine 객체를 가지질 수 있다(has-a).

상속과 구성

p571.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
```

Dog 클래스와 Animal 클래스의 관계는 is-a

Person 클래스와 Dog 클래스의 관계는 has-a
(실제 객체가 생성되어 대입)

Lab: Card와 Deck

- has-a 관계. 예제
- 카드를 나타내는 Card 클래스를 작성하고 52개의 Card 객체를 가지고 있는 Deck 클래스를 작성한다. 각 클래스의 __str__() 메소드를 구현하여서 덱 안에 들어 있는 카드를 출력한다.

p573.py

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

    #생성자로 카드 1장을 나타내는 객체를 초기화한다.
    #suit는 카드의 무늬, rank는 카드의 숫자이다.
    def __init__(self, suit, rank):
        self.suit = suit
        self.rank = rank


…
```

Lab: 학생과 강사

- 상속. 예제
- 일반적인 사람을 나타내는 Person 클래스를 정의한다. Person 클래스를 상속받아서 학생을 나타내는 클래스 Student와 선생님을 나타내는 클래스 Teacher를 정의한다. 모든 클래스에 __str__()을 정의하여 객체를 print() 함수로 출력하면 인스턴스 변수 값들이 출력되도록 하라

- Person 클래스

  name(이름), number(주민번호)
- Student 클래스

  name(이름), number(주민번호), classes(수강과목), gpa(평점)
- Teacher 클래스

  name(이름), number(주민번호), courses(강의과목), salary(월급)

Lab: 학생과 강사

p574.py

```
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)

…
```

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

이름=김철수
주민번호=123456790
강의과목=['Python']
월급=3000000

Lab: 게임에서의 상속

- 경기자가 미사일을 쏴서 적을 파괴하는 게임을 가정하자. 미사일이나 경기자를 Alien 클래스와 Player 클래스로 나타내려고 한다. 미사일이나 경기자는 공통부분을 가지고 있다. 이것들은 모두 움직이는 작은 그림이다. 이것을 Sprite 클래스로 모델링하자. Sprite 클래스를 상속받아서 Alien클래스와 Player 클래스를 작성한다.
- speed, name은 추가 변수이고 move()는 메소드 오버라이딩한다

p576.py

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

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

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

Lab: tkinter 프로그램에서 상속

- tkinter 모듈의 Canvas 클래스를 상속받아서 3대의 자동차 클래스를 작성한다. 각 자동차들은 Canvas 클래스 위에 자신을 그리는 생성자와 자신을 움직이는 함수 movement()를 가지고 있다. 난수를 이용하여 자동차들이 경주하는 프로그램을 작성해보자.

p577.py

```
from tkinter import *
import random

# Canvas 클래스를 상속받아서 Car 클래스를 정의한다.
class Car(Canvas):
    def __init__(self, master = None):   #캔버스를 생성하고 자동차를 그린다.
        Canvas.__init__(self, master, width = 500, height = 70)
        self.master = master
        self.dx = 0
        self.dy = 0
        self.rectangle = self.create_rectangle(0, 5, 70, 35, fill = "black")
        self.oval = self.create_oval(5, 20, 25, 50, fill = "black")
        self.oval2 = self.create_oval(40, 20, 60, 50, fill = "black")
        self.pack()
        …
```

이번 장에서 배운 것

- 상속은 다른 클래스를 재사용하는 탁월한 방법이다. 객체와 객체간의 __ 관계가 성립된다면 상속을 이용하도록 하자.
- 상속을 사용하면 중복된 코드를 줄일 수 있다. 공통적인 코드는 부모 클래스로 모으도록 하자.
- 상속에서는 부모 클래스의 메소드를 자식 클래스가 재정의할 수 있다. 이것을 메소드 _____이라고 한다.