# 06장 클래스

## 6.1 클래스와 객체

- 클래스(class)는 똑같은 무엇인가를 계속해서 만들어 낼 수 있는 설계 도면이고 객체(object)란 클래스로 만든 피조물을 뜻한다.
- 예를들면 붕어빵틀은 클래스이고, 만들어진 붕어빵을 객체(혹은 인스턴스)라 할 수 있다.

[ch06/ex01_class.py]

```
class FishBread:
    pass

a = FishBread()
b = FishBread()

print(type(a))
print(type(b))
print(id(a))
print(id(b))

'''
<class '__main__.FishBread'>
<class '__main__.FishBread'>
4220520
4420160
'''
```

## 6.2 클래스 선언

- 일반적으로 클래스는 클래스 변수(데이터)와  클래스 함수(메서드)로 구성된다.

```
# 클래스의 구조
class 클래스 이름[(상속 클래스명)]:
   <클래스 변수1>
   <클래스 변수2>
   ...
   <클래스 변수n>

   def 클래스 함수1(self[,인수1,인수2,,,]):
```

```
    <수행할 문장1>
    <수행할 문장2>
    ...
```

[ch06/ex02_declare.py]

```
result1 = 0
result2 = 0

# add 함수를 정의한다.
def add(num):
    global result1
    result1 += num  # result = result + num
    return result1
def sub(num):
    global result1
    result1 -= num
    return result1

print(add(3))  # 3
print(add(4))  # 7
print(sub(2))  # 5

def add2(num):
    global result2
    result2 += num
    return result2
def sub2(num):
    global result2
    result2 -= num
    return result2

print(add2(3))  # 3
print(add2(7))  # 10
print(sub2(2))  # 8


# 클래스 선언
class Calculator:
    def __init__(self):
        self.result = 0
    def add(self, num):
        self.result += num
        return self.result

cal1 = Calculator()
cal2 = Calculator()

print(cal1.add(3))  # 3
print(cal1.add(4))  # 7
print(cal2.add(3))  # 3
print(cal2.add(7))  # 10
```

[ch06/ex02_declare2.py]

```
#-*- coding:utf-8 -*-

class MyClass:
    """아주 간단한 클래스"""
    pass
print(dir())  #생성된 이름공간의 확인
print(type(MyClass))
```

```
class Person:
    name = "내 이름은 김연아!!"
    def Print(self):
        print("My Name is {0}".format(self.name))
p1 = Person()
p1.Print()          # 바운드 메서드 호출 방식
Person.Print(p1)    # 언바운드 메서드 호출 방식

'''
['MyClass', '__annotations__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__']
<class 'type'>
My Name is 내 이름은 김연아!!
My Name is 내 이름은 김연아!!
'''
```

## 6.3 사칙연산 클래스 만들기

[ch06/ex03_fourCal.py]

```
# 클래스 구조 만들기
class FourCal:
    def setdata(self, first, second):
        self.first = first
        self.second = second
    def add(self):
        result = self.first + self.second
        return result
    def mul(self):
        result = self.first * self.second
        return result
    def sub(self):
        result = self.first - self.second
        return result
    def div(self):
        result = self.first / self.second
        return result

a = FourCal()
print(type(a))  # 객체 a의 타입은 FourCal 클래스이다.
# <class '__main__.FourCal'>


# 객체에 숫자 지정할 수 있게 만들기
a.setdata(4, 2)
print(a.first)
# 4
print(a.second)
# 2

b = FourCal()
b.setdata(3, 7)
print(b.first)
# 3

print(id(a.first))  # a의 first 주소 값을 확인
# 8791524357040
print(id(b.first))  # b의 first 주소 값을 확인
# 8791524357008
```

```
# 곱하기, 빼기, 나누기 기능 만들기
print(a.add())
print(a.mul())
print(a.sub())
print(a.div())
print(b.add())
print(b.mul())
print(b.sub())
print(b.div())


'''
6
8
2
2.0
10
21
-4
0.42857142857142855
'''
```

## 6.4 생성자(Constructor)

■ 생성자 메서드는 인스턴스 객체가 생성될 때 자동으로 호출된다. 또한 생성자를 통해 인스턴스 변수에 초기값을 설정할 수 있다.

[ch06/ex04_constructor.py]

```
class FourCal:
    # 생성자 메서드: 1.객체를 생성한다.
    #               2.변수를 초기화한다.
    def __init__(self, first, second): # 생성자 메서드
        self.first = first
        self.second = second
    def setdata(self, first, second):
        self.first = first   # 객체 변수
        self.second = second # 객체 변수
    def add(self):  # 클래스 메서드(method) = 함수(function)
        result = self.first + self.second
        return result
    def mul(self):
        result = self.first * self.second
        return result
    def sub(self):
        result = self.first - self.second
        return result
    def div(self):
        result = self.first / self.second
        return result

#a = FourCal()
#a.add()  # 오류 발생, AttributeError: 'FourCal' object has no attribute 'first'

a = FourCal(4, 2)
print(a.first)
# 4
print(a.second)
# 2
print(a.add())
# 6
```

```
print(a.div())
# 2.0
```

**[과제] 메서드 추가하기**

- 객체변수 value가 100 이상의 값은 가질 수 없도록 제한하는 MaxLimitCalculator 클래스를 만들어 보자.

[ch06/verify.ve01_maxLimitCalculator.py]

```
#-*- coding:utf-8 -*-

class Calculator:
    def __init__(self):
        self.value = 0
    def add(self, val):
        self.value += val

# 여기에 코드를 기입한다.




cal = MaxLimitCalculator()
cal.add(50)
cal.add(60)

print(cal.value)
# 100
```

# 6.5 클래스의 상속

## 6.5.1 상속이란?

- 상속(Inheritance)을 이용하면 부모 클래스의 모든 속성(데이터, 메서드)을 자식 클래스로 물려줄 수 있다.

```
# 클래스의 상속
class 클래스명(상속할 클래스명):
```

[ch06/ex05_inheritance.py]

```
class FourCal:
    # 생성자 메서드: 1.객체를 생성한다.
    #               2.변수를 초기화한다.
    def __init__(self, first, second): # 생성자 메서드
        self.first = first
        self.second = second
    def setdata(self, first, second):
        self.first = first   # 클래스 변수
        self.second = second # 클래스 변수
    def add(self):  # 클래스 메서드(method) = 함수(function)
        result = self.first + self.second
```

```
        return result
    def mul(self):
        result = self.first * self.second
        return result
    def sub(self):
        result = self.first - self.second
        return result
    def div(self):
        result = self.first / self.second
        return result


# 클래스의 상속
class MoreFourCal(FourCal):
    def pow(self):
        result = self.first ** self.second
        return result

a = MoreFourCal(4, 2)
print(a.add()) # 6
print(a.pow()) # 4^2 = 16
```

~~[ch06/ex05_inheritance_1.py]~~

```
# 상속이란?
class Person:
    " 부모 클래스 "
    def __init__(self, name, phoneNumber):
        self.Name = name
        self.PhoneNumber = phoneNumber

    def PrintInfo(self):
        print("Info(Name:{0}, Phone Number: {1})".format(self.Name, self.PhoneNumber))

    def PrintPersonData(self):
        print("Person(Name:{0}, Phone Number: {1})".format(self.Name, self.PhoneNumber))

class Student(Person):
    " 자식 클래스 "
    def __init__(self, name, phoneNumber, subject, studentID):
        self.Name = name
        self.PhoneNumber = phoneNumber
        self.Subject = subject
        self.StudentID = studentID

p = Person("Derick", "010-123-4567")
s = Student("Marry", "010-654-1234", "Computer Science", "990999")
print(p.__dict__)  # 클래스의 정보는 내부적으로 __dict__라는 이름의 딕셔너리 객체로 관리된다.
print(s.__dict__)  # Student 인스턴스 객체

'''
{'Name': 'Derick', 'PhoneNumber': '010-123-4567'}
{'Name': 'Marry', 'PhoneNumber': '010-654-1234', 'Subject': 'Computer Science', 'StudentID': '990999'}
'''
```

## 6.5.2 클래스 간의 관계 확인

- 상속 관계인 두 클래스 간의 관계를 확인하기 위해 issubclass() 내장 함수를 이용할 수 있다.

[ch06/ex05_issubclass.py]

```
class Person:
    " 부모 클래스 "
    pass

class Student(Person):
    " 자식 클래스 "
    pass

# 클래스 간의 관계 확인
print(issubclass(Student, Person))  # True
print(issubclass(Person, Student))  # False
print(issubclass(Person, Person))   # True
print(issubclass(Person, object))   # True
print(issubclass(Student, object))  # True

class Dog:
    pass

print(issubclass(Student, Dog))  # False
print(issubclass(Dog, Person))   # False
print(issubclass(Dog, object))   # True
```

## 6.5.3 부모 클래스의 생성자 호출

■ 멤버 변수인 Name, PhoneNumber는 Person 클래스에 정의돼 있는 것으로 중복된 코드이다. 이 부분을 부모 클래스인 Person의 생성자를 호출하는 것으로 수정하면 다음과 같다.

[ch06/ex05_subclass.py]

```
# 부모 클래스의 생성자 호출
class Person:
    " 부모 클래스 "
    def __init__(self, name, phoneNumber):
        self.Name = name
        self.PhoneNumber = phoneNumber

    def PrintInfo(self):
        print("Info(Name:{0}, Phone Number: {1})".format(self.Name, self.PhoneNumber))

    def PrintPersonData(self):
        print("Person(Name:{0}, Phone Number: {1})".format(self.Name, self.PhoneNumber))

class Student(Person):
    " 자식 클래스 "
    def __init__(self, name, phoneNumber, subject, studentID):
        Person.__init__(self, name, phoneNumber)  # 명시적으로 Person 생성자를 호출
        #self.Name = name
        #self.PhoneNumber = phoneNumber
        self.Subject = subject
        self.StudentID = studentID

p = Person("Derick", "010-123-4567")
s = Student("Marry", "010-654-1234", "Computer Science", "990999")
print(p.__dict__)  # 클래스의 정보는 내부적으로 __dict__라는 이름의 딕셔너리 객체로 관리된다.
print(s.__dict__)  # Student 인스턴스 객체

'''
```

```
{'Name': 'Derick', 'PhoneNumber': '010-123-4567'}
{'Name': 'Marry', 'PhoneNumber': '010-654-1234', 'Subject': 'Computer Science', 'StudentID': '990999'}
'''
```

## 6.5.4 메서드 추가하기

■ 부모 클래스를 상속받은 자식 클래스는 멤버 변수와 멤버 메서드를 모두 상속받게 된다. 여기에 추가적인 기능이 필요한 경우, 자식 클래스에 메서드를 추가할 수 있다.

[ch06/ex05_methodAdd.py]

```
# 메서드 추가하기
class Person:
    " 부모 클래스 "
    def __init__(self, name, phoneNumber):
        self.Name = name
        self.PhoneNumber = phoneNumber

    def PrintInfo(self):
        print("Info(Name:{0}, Phone Number: {1})".format(self.Name, self.PhoneNumber))

    def PrintPersonData(self):
        print("Person(Name:{0}, Phone Number: {1})".format(self.Name, self.PhoneNumber))

class Student(Person):
    " 자식 클래스 "
    def __init__(self, name, phoneNumber, subject, studentID):
        Person.__init__(self, name, phoneNumber)  # 명시적으로 Person 생성자를 호출
        self.Subject = subject
        self.StudentID = studentID

    def PrintStudentData(self):  # 새로운 메서드를 추가
        print("Student(Subject: {0}, Studnet ID: {1})".format(self.Subject, self.StudentID))

s = Student("Marry", "010-654-1234", "Computer Science", "990999")
s.PrintPersonData()
s.PrintStudentData()

'''
Person(Name:Marry, Phone Number: 010-654-1234)
Student(Subject: Computer Science, Studnet ID: 990999)
'''
```

### [과제] 메서드 추가하기

■ 다음은 Calculator 클래스이다. 이 클래스를 상속하여 UpgradeCalculator를 만들고 값을 뺄 수 있는 minus 메서드를 추가해 보자.

[ch06/verify.ve02_calculator.py]

```
# Calculator 클래스를 상속하여 UpgradeCalculator를 만들자.

class Calculator:
    def __init__(self):
        self.value = 0
    def add(self, val):
        self.value += val
```

```
# 여기에 코드를 기입한다.



cal = UpgradeCalculator()
cal.add(10)
cal.minus(7)

print(cal.value)  # 10에서 7을 뺀 3을 출력
# 3
```

## 6.5.5 메서드 오버라이딩

■ 부모 클래스의 메서드에 대해 자식 클래스에서 재정의하는 것이다.

[ch06/ex05_methodOverriding.py]

```
class FourCal:
    # 생성자 메서드: 1.객체를 생성한다.
    #               2.변수를 초기화한다.
    def __init__(self, first, second): # 생성자 메서드
        self.first = first
        self.second = second
    def setdata(self, first, second):
        self.first = first   # 클래스 변수
        self.second = second # 클래스 변수
    def add(self):  # 클래스 메서드(method) = 함수(function)
        result = self.first + self.second
        return result
    def mul(self):
        result = self.first * self.second
        return result
    def sub(self):
        result = self.first - self.second
        return result
    def div(self):
        result = self.first / self.second
        return result


a = FourCal(4, 0)
#print(a.div())
# ZeroDivisionError: division by zero


class SafeFourCal(FourCal):
    # div 메서드를 재정의한다. (method overriding)
    def div(self):
        if self.second == 0:
            return 0
        else:
            return self.first / self.second

a = SafeFourCal(4, 0)
print(a.div())
# 0
```

~~[ch06/ex05_methodOverriding_1.py]~~

```
# 메서드 오버라이딩
class Person:
    " 부모 클래스 "
    def __init__(self, name, phoneNumber):
        self.Name = name
        self.PhoneNumber = phoneNumber

    def PrintInfo(self):
        print("Info(Name:{0}, Phone Number: {1})".format(self.Name, self.PhoneNumber))

    def PrintPersonData(self):
        print("Person(Name:{0}, Phone Number: {1})".format(self.Name, self.PhoneNumber))

class Student(Person):
    " 자식 클래스 "
    def __init__(self, name, phoneNumber, subject, studentID):
        Person.__init__(self, name, phoneNumber)  # 명시적으로 Person 생성자를 호출
        self.Subject = subject
        self.StudentID = studentID

    def PrintStudentData(self):  # 새로운 메서드를 추가
        print("Student(Subject: {0}, Studnet ID: {1})".format(self.Subject, self.StudentID))

    def PrintInfo(self):  # Person의 PrintInfo() 메서드를 재정의
        print("Info(Name:{0}, Phone Number: {1})".format(self.Name, self.PhoneNumber))
        print("Info(Subject: {0}, Student ID: {1}".format(self.Subject, self.StudentID))

s = Student("Marry", "010-654-1234", "Computer Science", "990999")
s.PrintInfo()  # 재정의된 메서드를 호출

'''
Info(Name:Marry, Phone Number: 010-654-1234)
Info(Subject: Computer Science, Student ID: 990999
'''
```

## ~~6.5.6 클래스 상속과 이름공간~~

- 상속 관계 검색의 원칙(Principles of the inheritance search)
  - 인스턴스 객체 영역 > 클래스 객체 간 상속을 통한 영역(자식 클래스 영역) > 부모 클래스 영역 > 전역 영역
- 자식 클래스가 상속받은 멤버 메서드에 대해 재정의하지 않거나 멤버 데이터에 새로운 값을 항당하지 않은 경우, 자식 클래스 내부의 이름공간에 해당 데이터와 메서드를 위한 저장 공간을 생성하지 않고 대신 단순히 부모 클래스의 이름공간에 존재하는 데이터와 메서드를 참조한다.

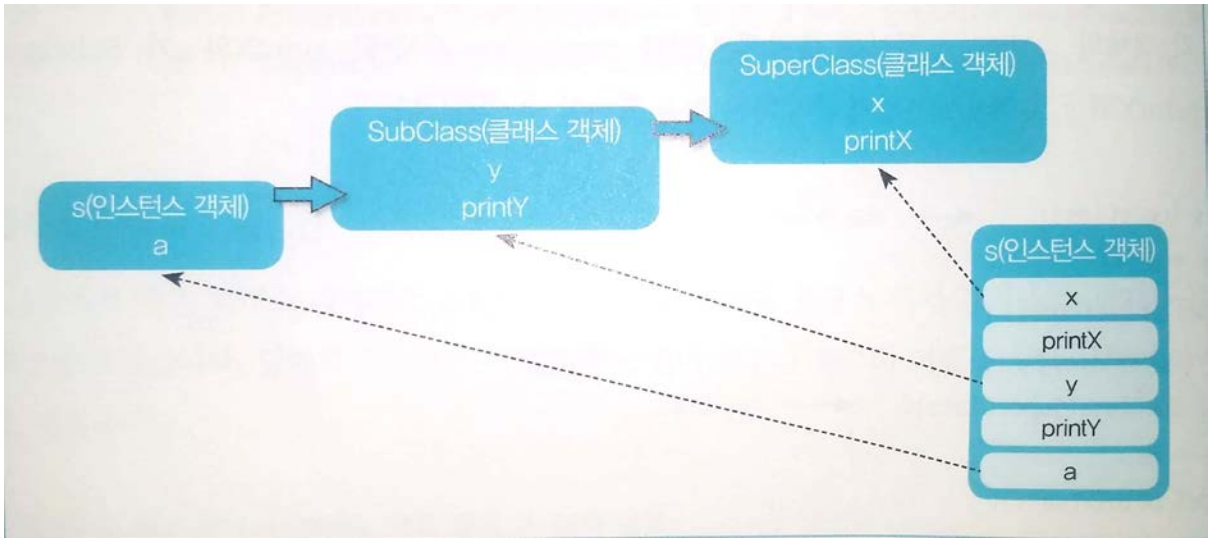
```
# 클래스 상속과 이름공간
class SuperClass:
    x = 10
    def printX(self):
        print(self.x)

class SubClass(SuperClass):
    y = 20
    def printY(self):
        print(self.y)

s = SubClass()
s.a = 30  # 인스턴스 객체 s에 멤버 변수 a 정의
```

```
# 클래스 객체와 인스턴스 객체의 이름 정보는 내부 변수 __dict__에 딕셔너리 형식으로 관리된다.
print("superClass: ", SuperClass.__dict__)
# superClass:  {'__module__': '__main__', 'x': 10, 'printX': <function SuperClass.printX at
0x00000000022957B8>, '__dict__': <attribute '__dict__' of 'SuperClass' objects>, '__weakref__': <attribute
'__weakref__' of 'SuperClass' objects>, '__doc__': None}
print("SubClass: ", SubClass.__dict__)
# SubClass:  {'__module__': '__main__', 'y': 20, 'printY': <function SubClass.printY at 0x00000000025E5AE8>,
'__doc__': None}
print("s: ", s.__dict__)
# s:  {'a': 30}
```

**[그림] 상속 관계에 대한 이름공간 도식화**




## ~~6.5.7 다중 상속~~

- 다중 상속이란 2개 이상의 클래스를 상속받는 경우를 말한다.
- 다양한 상속 구조에서 메서드의 이름을 찾는 순서는 __mro__에 튜플로 정의돼 있다.

```
# 다중 상속
class Tiger:
    def Jump(self):
        print("호랑이처럼 멀리 점프하기")
    def Cry(self):
        print("호랑이: 어흥~")

class Lion:
    def Bite(self):
        print("사자처럼 한입에 꿀꺽하기")
    def Cry(self):
        print("사장: 으르렁~")

class Liger(Tiger, Lion):  # Tiger 클래스, Lion 클래스 순서로 상속
    def Play(self):
        print("라이커만의 사육사와 재미있게 놀기")

l = Liger()
```

```
l.Bite()  # Lion 메서드 호출
l.Jump()  # Tiger 메서드 호출
l.Play()  # Liger 메서드 호출
l.Cry()   # Tiger 메서드 호출
print(Liger.__mro__)
'''
사자처럼 한입에 꿀꺽하기
호랑이처럼 멀리 점프하기
라이커만의 사육사와 재미있게 놀기
호랑이: 어흥~
(<class '__main__.Liger'>, <class '__main__.Tiger'>, <class '__main__.Lion'>, <class 'object'>)
'''
```

### ~~6.5.8 super()를 이용한 상위 클래스의 메서드 호출~~

- super() 내장 함수의 반환값은 부모 클래스의 객체를 반환하게 되며, 자바의 super(), C#의 base 키워드와 유사하다.
- 생성자 호출 순서는 MRO()의 역순으로 상위 클래스부터 호출된다.

```
# super()를 이용한 상위 클래스의 메서드 호출

class Animal:
    def __init__(self):
        print("Animal __init__()")

class Tiger(Animal):
    def __init__(self):
        super().__init__()  # 부모 클래스의 생성자 메서드 호출
        print("Tiger __init_()")

class Lion(Animal):
    def __init__(self):
        super().__init__()  # 부모 클래스의 생성자 메서드 호출
        print("Lion __init_()")

class Liger(Tiger, Lion):
    def __init__(self):
        super().__init__()  # 부모 클래스의 생성자 메서드 호출
        print("Liger __init_()")

l = Liger()
'''
Animal __init__()
Lion __init_()
Tiger __init_()
Liger __init_()
'''
```

## 6.6 클래스 변수

- 객체변수는 다른 객체들에 영향받지 않고 독립적으로 그 값을 유지한다.
- 클래스 변수는 클래스 안에 함수를 선언하는 것과 마찬가지로 클래스 안에 변수를 선언하여 생성한다.

[ch06/ex05_classVariable.py]

```
# 클래스 변수는 클래스로 만든 모든 객체에 공유된다.
class Family:
    lastname = "김"  # 클래스 변수

print(Family.lastname)  # 김

a = Family()
b = Family()
print(a.lastname)  # 김
print(b.lastname)  # 김

Family.lastname = "박"
b.lastname = "이"
print(a.lastname)  # 박
print(b.lastname)  # 이

# id 함수는 객체의 주소값를 바탕으로 계산된 정수값이다.
print(id(Family.lastname))
print(id(a.lastname))
print(id(b.lastname))
'''
7131008
7131008
32284800
'''
```

## 6.7 연산자 오버로딩

■ 연산자 오버로딩(Overloading)이란 연산자(+,-,*,/…)를 객체끼리 사용할 수 있게 하는 기법이다.

```
class HousePark:
    lastname = "박"
    def __init__(self,name):  # 생성자
        self.fullname = self.lastname + name
    def travel(self,where):
        print("%s, %s여행을 가다." %(self.fullname,where))
    def love(self,other):
        print("%s, %s 사랑에 빠졌네." %(self.fullname, other.fullname))
    def __add__(self, other):  # '+' 연산자 오버로딩 정의
        print("%s, %s 결혼했네." %(self.fullname, other.fullname))

class HouseKim(HousePark):
    lastname = "김"
    def travel(self,where,day):
        print("%s, %s여행 %d일 가네." %(self.fullname,where,day))

pey = HousePark("응용")
juliet = HouseKim("줄리엣")
pey.love(juliet)
pey + juliet
```

## [과제] 연습문제

Q1 다음과 같이 동작하는 클래스 Calculator를 작성해 보자.

```
cal1 = Calculator([1,2,3,4,5])
print(cal1.sum())
#15
print(cal1.avg())
#3.0

cal2 = Calculator([6,7,8,9,10])
print(cal2.sum())
#40
print(cal2.avg())
#8.0
```