파이썬 클래스 기본

클래스와 객체(인스턴스) (1)

객체(Object)

- 데이터(속성)와 코드, 함수(메서드)를 묶어서 조직화 하는 방법

```
[자동차- 객체]
- 메이커, 모델, 색상, 연식, 가격 등-> 속성(Attribute), 변수
- 시동걸기, 가속하기, 주행하기, 주차하기 등 -> 동작 (Action)
```

클래스란

- 클래스는 객체 지향 프로그램(OOP, Object Oriented Programming)의 기본단위이면, 함수와 변수가 포함될 수 있다.
- 객체를 정의해 놓은 것으로, 객체의 설계도 또는 틀에 해당
- 클래스는 객체를 생성하는데 사용된다

클래스와 객체(인스턴스) (2)

클래스와 객체의 관계

- 클래스가 제품을 찍어내는 틀이라면, 객체는 틀에서 만들어져 나오는 제품이다.
- 클래스는 객체를 생성하는데 사용될 뿐이고, 객체 그 자체는 아니다.
- 클래스를 한번만 잘 만들어 놓으면 객체는 클래스로부터 계속 생성될 수 있다.

객체와 인스턴스

- 클래스로부터 만들어진 객체를 인스턴스라고 함
- 인스턴스와 객체는 같은 의미임
- 객체는 모든 인스턴스를 대표하는 포괄적 의미가 있고
- 인스턴스는 어떤 클래스로부터 만들어진 객체인지 강조하는 구체적인 의미가 있다.
- 인스턴스 객체는 각각 독립적이며, 서로 영향을 주지 않는다.

클래스와 객체(인스턴스) (3)

절차지향언어

```
# 사각형 1
h1=4
v1=5
area1=h1*v1
print(area1)

# 사각형 2
h2=10
v2=50
area2=h2*v2
print(area2)
```

```
20
500
```

```
def area(h,v):
    return h*v

#사각형 1
h2=4
v2=5
print(area(h2,v2))

#사각형 2
h3=10
v3=50
print(area(h3,v3))
```

```
20
500
```

객체지향 언어

```
class Rectangle:
    def __init__(self, h, v):
        self.h=h
        self.v=v
    def area(self):
        return self.h*self.v

a=Rectangle(4,5)   #사각형1
b=Rectangle(10,50) #사각형2

# 각 사각형의 넓이, 가로, 세로 길이 확인
print(a.area())
print(b.area())
print(a.h)
print(a.v)
print(b.h)
print(b.v)
b.h=0
print(b.area())
print(b.h)
```

```
20
500
4
5
10
50
0
0
```

클래스 속성과 인스턴스 속성

클래스 속성(Class Attribute)

- 클래스 속성은 클래스 정의 내부에 정의되며, 모든 인스턴스에 대해 공유됨

```
class 클래스명:
    속성 = 값
```

- 즉, 해당 클래스의 모든 인스턴스 간에 같은 값을 가지게 됨
- 클래스 속성에 접근하기 위해서는 **클래스이름.속성이름**을 사용

인스턴스 속성(Instance Attribute)

- 인스턴스 속성은 클래스의 각 인스턴스(객체)에 속하며, 각 인스턴스마다 독립적인 값을 가짐
- 인스턴스 속성은 **__init__ 메서드(생성자)** 내에서 **self.속성이름** 형태로 정의됨

```
class 클래스명:
    def __init__(self):
        self.속성=값 #인스턴스 속성 값 만들기
```

- 각 인스턴스는 자신만의 별도의 인스턴스 속성을 가지게 되므로, 동일한 클래스의 다른 인스턴스와는 독립적인 값을 유지함
- 인스턴스 속성에 접근하기 위해서는 **인스턴스명.속성이름**을 사용

클래스 속성과 인스턴스 속성: 클래스 속성(1)

클래스 속성(Class Attribute)

1
```
# 클래스 선언
class A:
    n=3 #클래스 변수 지정

# 인스턴스 생성: 인스턴스 = 클래스명()
a=A()
b=A()
```

2
```
# 각 네임스페이스 확인: 객체.__dict__
print('------네임스페이스 확인--------')
print('A.__dict__',A.__dict__) # A클래스 네임스페이스 확인
print('a.__dict__',a.__dict__) # a인스턴스 네임스페이스 확인
print('b.__dict__',b.__dict__) # b인스턴스 네임스페이스 확인
```

```
------네임스페이스 확인--------
A.__dict__ {'__module__': '__main__', 'n': 3, '__dict__': <attrib
_' of 'A' objects>, '__doc__': None}
a.__dict__ {}
b.__dict__ {}
```

A클래스의 네임스페이스에 ' n ' :3이 저장되어 있고,
a, b 각 인스턴스의 네임스페이스는 비어 있음

3
```
# 클래스 변수 확인
print('------클래스 변수확인--------')
print('A.n:', A.n) # 클래스 변수 접근
print('a.n:', a.n) # a인스턴스를 통해 클래스 변수 접근
print('b.n:', b.n) # b인스턴스를 통해 클래스 변수 접근
```

```
------클래스 변수확인--------
A.n: 3
a.n: 3
b.n: 3
```

A.n은 A클래스의 네임스페이스에 있는 n의 값 3이 출력 됨
a.n, b.n 각 인스턴스의 네임스페이스에 n이 없으므로,
클래스변수 n의 값 3을 출력함

클래스 속성과 인스턴스 속성: 클래스 속성(2)

4
```
# 클래스 변수 변경
print('------ A.n= 100 :: 클래스 변수 변경---------')
A.n= 100 #a인스턴스에 변수 할당
print('A.n:', A.n) # 클래스 변수 접근
print('a.n:', a.n) # a인스턴스를 통해 클래스 변수 접근
print('b.n:', b.n) # b인스턴스를 통해 클래스 변수 접근
```

```
------ A.n= 100 :: 클래스 변수 변경---------
A.n: 100
a.n: 100
b.n: 100
```

A.n=100으로 클래스 변수 n의 값을 100으로 재할당

a.n, b.n 각 인스턴스에 클래스 변수 n의 값 100이 공유됨

5
```
# a.n 변수 할당
print('------a.n=50:: 인스턴스 변수 할당----------')
a.n = 50
print('A.n:', A.n)
print('a.n:', a.n)
print('b.n:', b.n)
```

```
------a.n=50:: 인스턴스 변수 할당----------
A.n: 100
a.n: 50
b.n: 100
```

a.n=50은 클래스변수 n의 값을 50으로 수정한 것 처럼 보이지만,

인스턴스a의 인스턴스변수 n=50으로 새로 생성된 것임

6
```
print('------네임스페이스 확인--------')
print('A.__dict__:',A.__dict__) # A클래스 네임스페이스 확인
print('a.__dict__:',a.__dict__) # a인스턴스 네임스페이스 확인
print('b.__dict__:',b.__dict__) # b인스턴스 네임스페이스 확인
```

각 객체의 네임스페이스를 확인할 수 있음

```
------네임스페이스 확인--------
A.__dict__: {'__module__': '__main__', 'n': 100, '__dict__'
f__' of 'A' objects>, '__doc__': None}
a.__dict__: {'n': 50}
b.__dict__: {}
```

클래스 속성과 인스턴스 속성: 인스턴스 속성(1)

인스턴스 속성(Instance Attribute)

1
```
#클래스 정의
class B:
    n=100 #클래스 변수
    def __init__(self,data):
        self.data=data #인스턴스 변수

    def show(self): #인스턴스 메소드
        print(f'클래스변수 B.n: {B.n}이고, 인스턴스변수  self.data: {self.data} 이다.')
```

def __init__(self, data) → 생성자 메서드

- **자동 호출:** 클래스의 새 인스턴스가 생성될 때 자동 호출 됨, 인스턴스 변수에 초기값 설정함
- **첫 번째 매개변수로 self를 사용:** self는 현재 인스턴스를 가리키는 참조자
- 모든 클래스에 __init__ 메서드를 정의할 필요없음. 만약 생략하면 파이썬은 기본적으로 빈 '__init__' 메서드 제공

def show(self) → 인스턴스 메서드

- 특정 인스턴스에 속한 메서드(함수), 첫 번째 매개변수로 반드시 'self'를 사용함 (self는 현재 인스턴스를 참조함)
- 클래스 외부에서 접근할 때는 객체.메서드( )로 실행
- 클래스 변수 접근 → 클래스명.속성, 인스턴스 변수 접근 → self.속성

클래스 속성과 인스턴스 속성: 인스턴스 속성(2)

1

```
#클래스 정의
class B:
    n=100 #클래스 변수
    def __init__(self,data):
        self.data=data #인스턴스 변수

    def show(self): #인스턴스 메소드
        print(f'클래스변수 B.n: {B.n}이고, 인스턴스변수  self.data: {self.data} 이다.')

#인스턴스 생성
b1=B(5)
b2=B(3)
```

인스턴스 생성

- **b1=B(5)**

  B 클래스의 인스턴스 b1을 생성

  이때 생성자 __init__(self, data)이 호출되고, self에는 b1, data에는 5가 전달됨

- **b2=B(3)**

  B 클래스의 인스턴스 b2을 생성

  이때 생성자 __init__(self, data)이 호출되고, self에는 b2, data에는 3이 전달됨

클래스 속성과 인스턴스 속성: 인스턴스 속성(3)

2

```
#네임스페이스 확인
print('--------네임스페이스 확인--------')
print('B.__dict__',B.__dict__)
print('b1.__dict__',b1.__dict__)
print('b2.__dict__',b2.__dict__)
```

```
--------네임스페이스 확인--------
B.__dict__ {'__module__': '__main__', 'n': 100, '__init__': <
0000027AD9A2C360>, '__dict__': <attribute '__dict__' of 'B' o
_': None}
b1.__dict__ {'data': 5}
b2.__dict__ {'data': 3}
```

객체명.__dict__ 명령으로 각 객체의 네임스페이스 확인

3

```
# 인스턴스변수, 클래스 변수 확인
print('--------인스턴스변수, 클래스 변수 확인--------')
print(f'B.n: {B.n}')
print(f'b1.n: {b1.n}, b1.data: {b1.data}')
print(f'b2.n: {b2.n}, b2.data: {b2.data}')
```

```
--------인스턴스변수, 클래스 변수 확인--------
B.n: 100
b1.n: 100, b1.data: 5
b2.n: 100, b2.data: 3
```

각 객체의 속성값을 확인 함

클래스 속성 → 클래스명.속성 (B.n)

인스턴스 속성 → 인스턴스.속성(b1.n, b2.n)

클래스 속성과 인스턴스 속성: 인스턴스 속성(3)

4

```
# 클래스 변수 변경
print('--------B.n=0::클래스 변수 변경--------')
B.n=0
print(f'B.n: {B.n}')
print(f'b1.n: {b1.n}, b1.data: {b1.data}')
print(f'b2.n: {b2.n}, b2.data: {b2.data}')
```

```
--------B.n=0::클래스 변수 변경--------
B.n: 0
b1.n: 0, b1.data: 5
b2.n: 0, b2.data: 3
```

B.n 클래스 변수값을 0으로 수정하면

모든 인스턴스 b1, b2의 n도 모두 0임을 확인

5

```
# 인스턴스 변수 변경
print('--------b1.data=0:: 인스턴스 변수 변경--------')
b1.data=0
print(f'B.n: {B.n}')
print(f'b1.n: {b1.n}, b1.data: {b1.data}')
print(f'b2.n: {b2.n}, b2.data: {b2.data}')
```

```
--------b1.data=0:: 인스턴스 변수 변경--------
B.n: 0
b1.n: 0, b1.data: 0
b2.n: 0, b2.data: 3
```

인스턴스 b1.n을 변경하면 해당 속성만 변경됨

6

```
# show()메소드 확인
print('--------show() 메소드 확인----------')
b1.show()
b2.show()
```

```
--------show() 메소드 확인----------
클래스변수 B.n: 0이고, 인스턴스변수  self.data: 0 이다.
클래스변수 B.n: 0이고, 인스턴스변수  self.data: 3 이다.
```

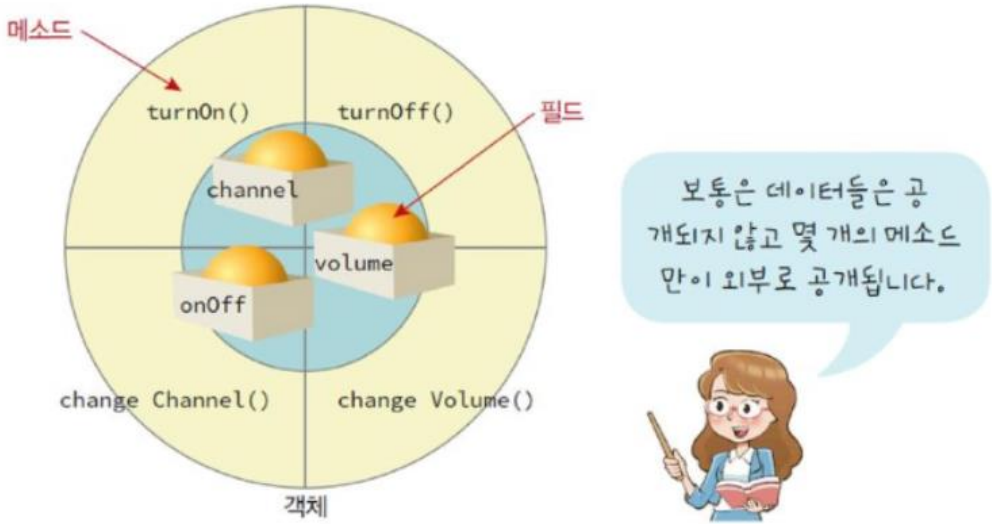
인스턴스.메서드( )

각 인스턴스별 show()메서드를 실행함

클래스 속성과 인스턴스 속성: 비공개(private) 속성(1)

비공개 속성(Private Attribute): 정보은닉

- 클래스 속성을 만들 때 **__속성**과 같이 __(밑줄 두 개)로 시작하면 비공개 속성이 됨
- 클래스 안에서만 접근할 수 있고, 클래스 바깥에서는 접근할 수 없어 클래스 안의 데이터를 외부에서 변경하지 못하게 함 (정보은닉)
- 클래스 외부에서는 클래스 변수를 바로 접근을 못하고, 메서드를 통해 사용하도록 함



사진출처:slidesplayer.org

클래스 속성과 인스턴스 속성: 비공개(private) 속성(2)

```
class Student:
    def __init__(self, name=None, age=0):  ❶
        self.__name=name
        self.__age=age

s1=Student()  ❷
print(s1.__name)  ❸
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[62], line 7
      4         self.__age=age
      6 s1=Student()
----> 7 print(s1.__name)

AttributeError: 'Student' object has no attribute '__name'
```

❶
- s1-Student() 명령으로 자동호출됨(생성자)
- name과 age는 기본값(None과 0)으로 설정
- __name과 __age는 비공개 속성값으로 클래스 외부에서 접근할 수 없음

❷
- s1-Student() 인스턴스 생성
- Student클래스의 __init__메서드 자동 실행
- name과 age값을 생략했으므로, name=None, age=0으로 값 전달

❸
- 비공개 속성 __name에 접근하려해서 'AttributeError'발생
- '_클래스명__속성'형태로 접근가능하나 권장사항 아님 s1._Student__name으로 접근 가능함

```
7 print(s1._Student__name)
```

```
None
```

클래스 속성과 인스턴스 속성: 비공개(private) 속성(3)

접근자와 설정자

- 클래스 외부에서 비공개 속성을 접근하려면 접근자(getters),설정자(setters)와 같은 메서드를 통해 접근함

❶
```
class Student:
    def __init__(self, name=None, age=0):
        self.__name=name
        self.__age=age

    def get_age(self):
        return self.__age

    def get_name(self):
        return self.__name

    def set_age(self, age):
        self.__age=age

    def set_name(self,name):
        self.__name=name
```

❷
```
s1=Student('홍길동',30)
print(s1.get_age())
s1.set_age(28)
print(s1.get_age())
```

```
30
28
```

❶ __name과 __age 인스턴스 변수가 비공개 속성으로 지정되어 있어
접근자(get_age, get_name)메서드를 정의하고
설정자(set_age, set_name) 메서드를 정의함

❷
- **s1=Student('홍길동',30)**
  : s1인스턴스 생성 - name은 '홍길동', age는 30 값을 전달
- **print(s1.get_age())**
  : get_age()메서드를 통해 s1인스턴스 age값을 반환
- **s1.set_age(28)**
  : set_age(28)메서드를 통해 s1인스턴스 age를 변경

클래스 속성과 인스턴스 속성: 비공개(private) 속성(3)

접근자와 설정자의 사용 이유

- 클래스 업그레이드할 때 편리함
- 접근자에서 매개변수를 통하여 잘못된 값을 사전에 차단할 수 있음
- 접근자만을 제공하면 읽기만 가능한 인스턴스 변수를 만들 수 있음

```
obj.age = -10
```

← 메서드를 통하지 않고 인스턴스 변수 age를 변경하면, 잘못된 값이 입력될 수 있음

```
def set_age(self, age):
    if age < 0:
        self.age = 0
    else:
        self.age = age
```

← 설정자를 통하면 잘못된 입력값을 사전에 차단할 수 있고,

클래스 업그레이드할 때도 편리함

클래스 메서드 (1)

메서드 정의

- 메서드: 클래스에 포함된 함수
- 형식

```
class 클래스이름:
    def 메소드(self, 변수):
        코드
```

메서드의 첫 번째 매개변수는 반드시 self로 지정

- 일반적인 함수와 똑같이 정의하지만 첫 번째 매개변수로 self를 사용
- self는 인스턴스 객체 자신의 레퍼런스를 지니고 있음

→ 각 인스턴스들은 self를 활용해 자신의 네임스페이스(이름공간)에 접근 가능

클래스 메서드 (2)

메서드 호출

1. 클래스 외부에서 호출

- 인스턴스 객체를 활용한 메서드 호출 → **인스턴스.메서드( )**
- 클래스 객체를 이용한 메서드 호출 → **클래스명.메서드(인스턴스,....) *첫번째 매개변수에 반드시 인스턴스로 지정**

```
class A:
    def set_value(self,v):
        self.value = v

    def get_value(self):
        return self.value

a=A()                   #인스턴스 생성
a.set_value(10)         # a인스턴스에서 메소드 호출
print(a.get_value())    # a인스턴스에서 메소드 호출

A.set_value(a,20)       #A클래스에서 메소드 호출
print(A.get_value(a)) #A클래스에서 메소드 호출
```

```
10
20
```

클래스 메서드 (3)

메서드 호출

2. 클래스 내부에서 호출

- self를 적어주면 클래스 내부에서 해당 메서드를 검색 → **self.메서드( )**
- self를 적어주지 않으면 클래스 외부에서 해당 메서드를 검색 → **메서드( )**

```
def set_value(i):
    print('외부 함수 호출:',i)
class A:
    def set_value(self,v):
        self.value=v

    def get_value(self):
        return self.value

    def vlaue_incr(self):
        self.set_value(self.value+1)

a = A() #인스턴스 생성
a.set_value(20)
print(a.get_value())
a.vlaue_incr()
print(a.get_value())
```

```
20
21
```

self가 있으므로 내부 메서드 호출

```
def set_value(i):
    print('외부 함수 호출:',i)
class A:
    def set_value(self,v):
        self.value=v

    def get_value(self):
        return self.value

    def vlaue_incr(self):
        set_value(self.value+1)

a = A() #인스턴스 생성
a.set_value(20)
print(a.get_value())
a.vlaue_incr()
print(a.get_value())
```

```
20
외부 함수 호출: 21
20
```

self가 없으므로 외부 함수 호출

클래스 메서드 – 정적 메서드(Static Method)

정적 메서드(Static Method)

- 특정 클래스에 속하지만, 클래스의 인스턴스(객체)가 아니라 클래스 자체에 바인딩된 메서드
- 정적 메서드는 인스턴스를 통해서가 아니라 클래스 이름을 통해 직접 호출
- 인스턴스를 통해 접근하지 않으므로, self 매개변수를 받지 않음
- 정적 메서드는 @staticmethod 데코레이터를 사용하여 정의됨

```
class Math:
    @staticmethod
    def add(x, y):
        return x + y

print(Math.add(5, 10))
m=Math()
print(m.add(5,10))
```

인스턴스 생성 없이 클래스에서 직접 호출

인스턴스 통해서도 호출 가능

```
15
15
```

클래스 메서드 – 클래스 메서드(Class Method)

클래스 메서드(Class Method)

- 클래스 메서드는 첫번째 인자로 cls를 받음, 이를 통해 클래스 내부에서 접근가능 → **cls.메서드( ) 또는 클래스명.메서드( )**
- @classmethod 데코레이터 사용하여 정의함
- 인스턴스 없이 클래스 이름을 사용하여 직접 호출할 수 있음 → **클래스명.클래스메서드( )**
- 클래스 메서드 클래스 변수에 접근하고 수정하는데 사용할 수 있음

```
class A:
    class_variable = 100   ←—— 클래스 변수 지정

    @classmethod
    def class_method(cls):  ←—— 클래스 메서드 작성
        return f"클래스 변수의 값은 {cls.class_variable}입니다."  ←—— 클래스내부에서 cls.클래스변수로 접근 함

print(A.class_method())  ←—— 인스턴스 생성없이 클래스명으로 직접 호출 가능
a=A()
print(a.class_method())  ←—— 인스턴스 통해서도 호출 가능
```

```
클래스 변수의 값은 100입니다.
클래스 변수의 값은 100입니다.
```

실습

실습1

```
## 클래스 연습
class Myclass:
    cnt=0    # 클래스 변수

    # 생성자(초기자)
    def __init__(self, 가로, 세로):
        self.width = 가로
        self.height = 세로
        Myclass.cnt += 1

    # 인스턴스 메서스
    def calarea(self):
        area = self.width * self.height
        return area

    def prncnt(self):
        print(Myclass.cnt)

a=Myclass(5,4)
b=Myclass(10,2)
print(a.calarea())
print(b.calarea())
a.prncnt()
b.prncnt()
print(Myclass.cnt)
```

결과

```
20
20
2
2
2
```

실습2

연습 문제 2.12.1

삼각형의 넓이를 계산하기 위한 클래스를 만든다. 이 클래스는 다음과 같은 속성을 가진다.

- 밑변의 길이 `b` 와 높이 `h`
- 삼각형의 넓이를 계산하는 메서드 `area`

```
class Triangle:
    def __init__(self, b, h):
        self.b = b
        self.h = h

    def area(self):
        return (self.b * self.h) / 2

# 사용 예
triangle = Triangle(3, 4)
print("넓이:", triangle.area())  # 넓이 출력
```

실습3

연습 문제 2.12.2

사각 기둥의 부피를 계산하기 위한 클래스를 만든다. 이 클래스는 다음과 같은 속성을 가진다.

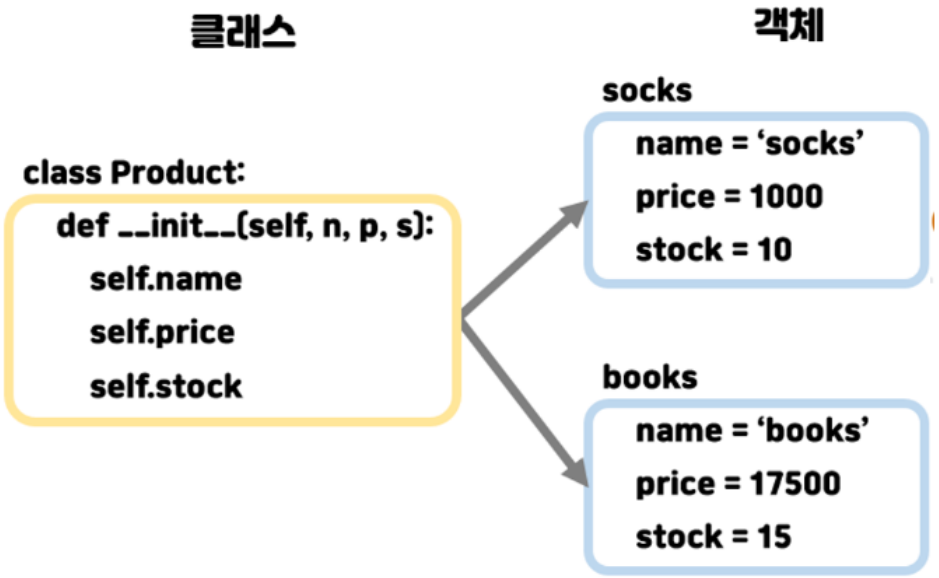
- 밑면의 가로 길이 `a`, 밑면의 세로 길이 `b`, 높이 `h`
- 부피를 계산하는 메서드 `volume`
- 겉넓이를 계산하는 메서드 `surface`

```
class RectangularPrism:
    def __init__(self, a, b, h):
        self.a = a
        self.b = b
        self.h = h

    def volume(self):
        return self.a * self.b * self.h

    def surface(self):
        return 2 * (self.a * self.b + self.a * self.h + self.b * self.h)

# 사용 예
prism = RectangularPrism(3, 4, 5)
print("부피:", prism.volume())  # 부피 출력
print("겉넓이:", prism.surface())  # 겉넓이 출력
```

온라인 쇼핑 프로그램 작성: 상품 클래스

상품 클래스

- 속성(Attributes): name(상품이름), price(가격), stock(재고)




```python
class Product:
    def __init__(self, n, p, s):
        self.name, self.price, self.stock = n, p, s

    def information(self):
        print('상품 이름 : ', self.name)
        print('상품 가격 : ', self.price)
        print('남은 재고 : ', self.stock)
```

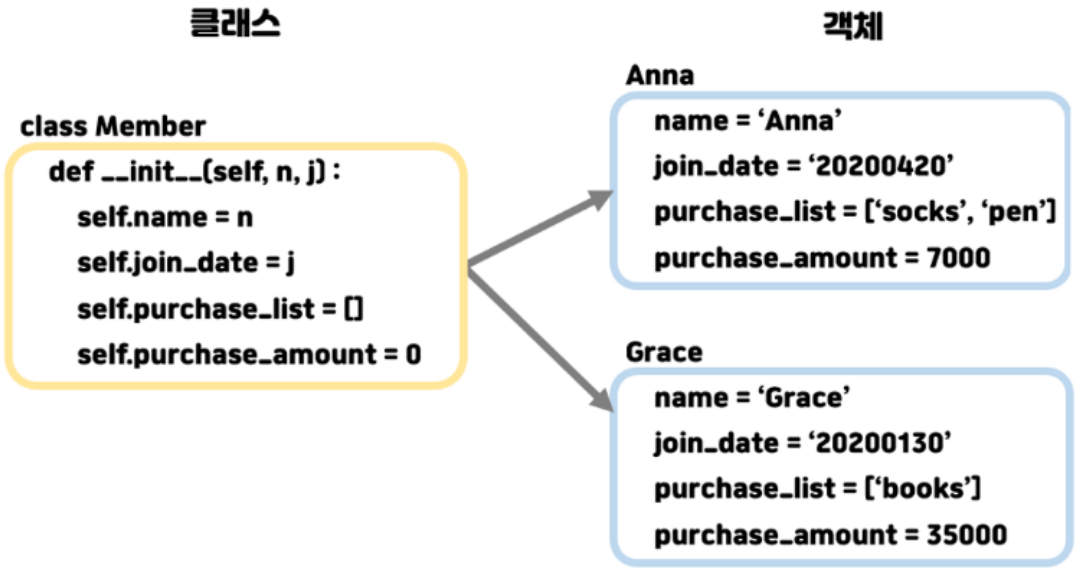
```python
# 물건 등록
socks = Product('socks', 1000, 10)
books = Product('books', 17500, 15)
```

온라인 쇼핑 프로그램 작성: 회원 클래스

회원 클래스

- 속성(Attributes):

  name(회원이름), join_date(가입날짜), purchase_list(구매내역), purchase_amount(누적 구매금액)




```python
class Member:
    def __init__(self, n, d):
        self.name, self.join_date = n, d
        self.purchase_list = []
        self.purchase_amount = 0

    def information(self):
        print('회원 이름 :', self.name)
        print('가입 날짜 :', self.join_date)
        print('구매 내역 :', self.purchase_list)
        print('누적 구매금액 :', self.purchase_amount)

    # Product 객체가 인자로 전달됨
    def buy(self, product, count):
        print(self.name, '고객님이', product.name, count, '개 구매!')
        product.stock -= count      # 구매 개수만큼 상품 재고 감소
        self.purchase_list.append(product.name) # 구매내역에 상품 추가
        self.purchase_amount += (product.price * count) # 누적 구매금액 수정
```

```python
# 회원가입
Anna = Member('Anna', '20200420')
Grace = Member('Grace', '20200130')
```

온라인 쇼핑 프로그램 작성 : 전체 코드

①

```
class Product:
    def __init__(self, n, p, s):
        self.name, self.price, self.stock = n, p, s

    def information(self):
        print('상품 이름 : ', self.name)
        print('상품 가격 : ', self.price)
        print('남은 재고 : ', self.stock)

class Member:
    def __init__(self, n, d):
        self.name, self.join_date = n, d
        self.purchase_list = []
        self.purchase_amount = 0

    def information(self):
        print('회원 이름 :', self.name)
        print('가입 날짜 :', self.join_date)
        print('구매 내역 :', self.purchase_list)
        print('누적 구매금액 :', self.purchase_amount)

    # Product 객체가 인자로 전달됨
    def buy(self, product, count):
        print(self.name, '고객님이', product.name, count, '개 구매!')
        product.stock -= count      # 구매 개수만큼 상품 재고 감소
        self.purchase_list.append(product.name) # 구매내역에 상품 추가
        self.purchase_amount += (product.price * count) # 누적 구매금액 수정
```

②

```
# 물건 등록
socks = Product('socks', 1000, 10)
books = Product('books', 17500, 15)

# 회원가입
Anna = Member('Anna', '20200420')
Grace = Member('Grace', '20200130')
```

③

```
Anna.buy(socks, 1)
```

```
Anna 고객님이 socks 1 개 구매!
```

```
Anna.information()
```

```
회원 이름 : Anna
가입 날짜 : 20200420
구매 내역 : ['socks']
누적 구매금액 : 1000
```

```
socks.information()
```

```
상품 이름 :  socks
상품 가격 :  1000
남은 재고 :  9
```

④

```
Grace.buy(books, 2)
```

```
Grace 고객님이 books 2 개 구매!
```

```
books.information()
```

```
상품 이름 :  books
상품 가격 :  17500
남은 재고 :  13
```

```
Grace.information()
```

```
회원 이름 : Grace
가입 날짜 : 20200130
구매 내역 : ['books']
누적 구매금액 : 35000
```

⑤

```
# 새로운 물건 입고
pen = Product('pen', 1200, 25)

Anna.buy(pen, 5)
```

```
Anna 고객님이 pen 5 개 구매!
```

```
Anna.information()
```

```
회원 이름 : Anna
가입 날짜 : 20200420
구매 내역 : ['socks', 'pen']
누적 구매금액 : 7000
```