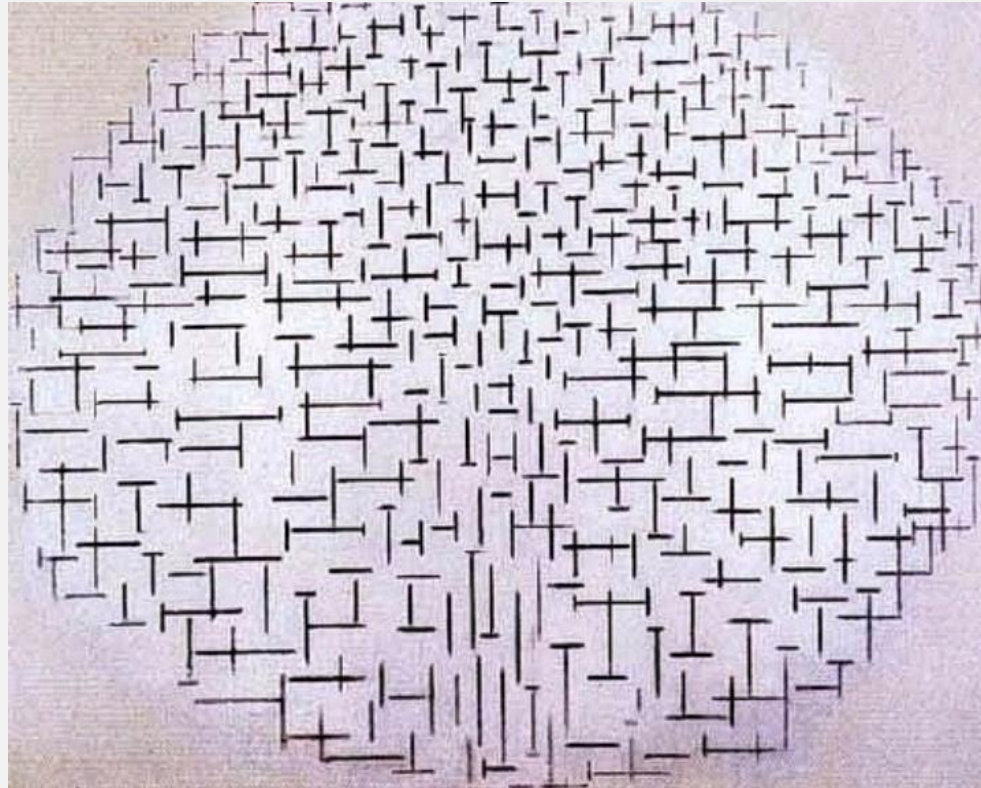


Class와 Object

By 박수현

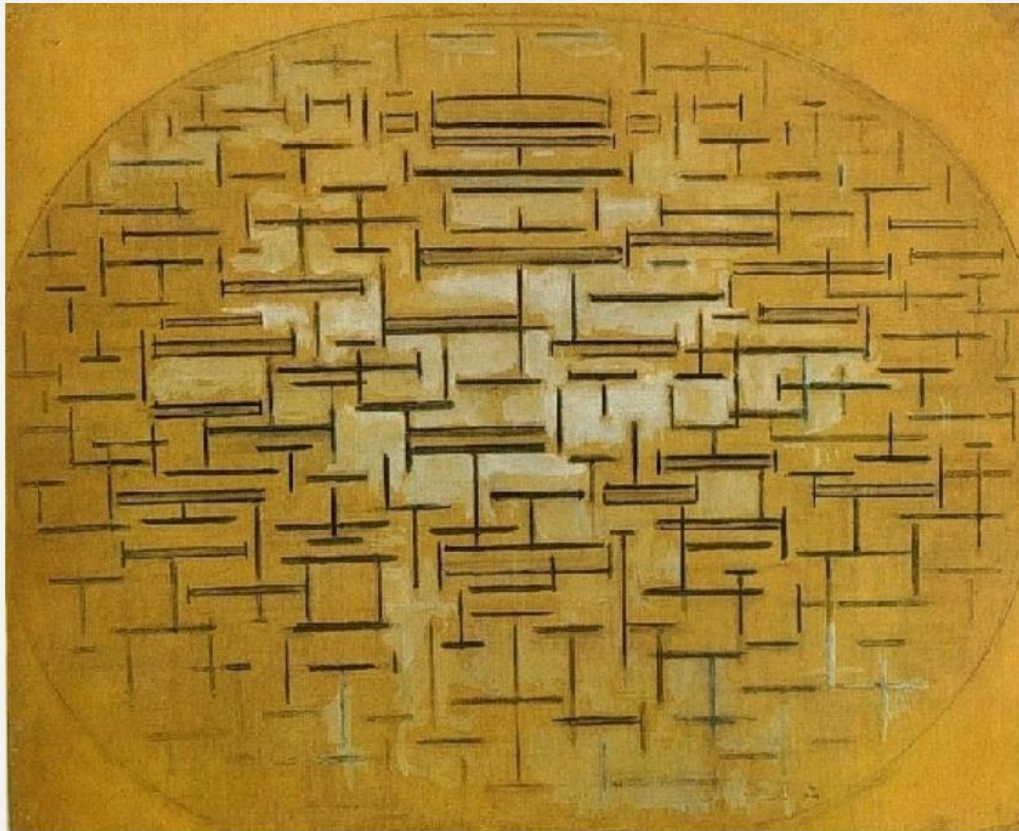
Programming Paradigm

- 다음 그림은 무엇을 그린 것일까 ?



Programming Paradigm

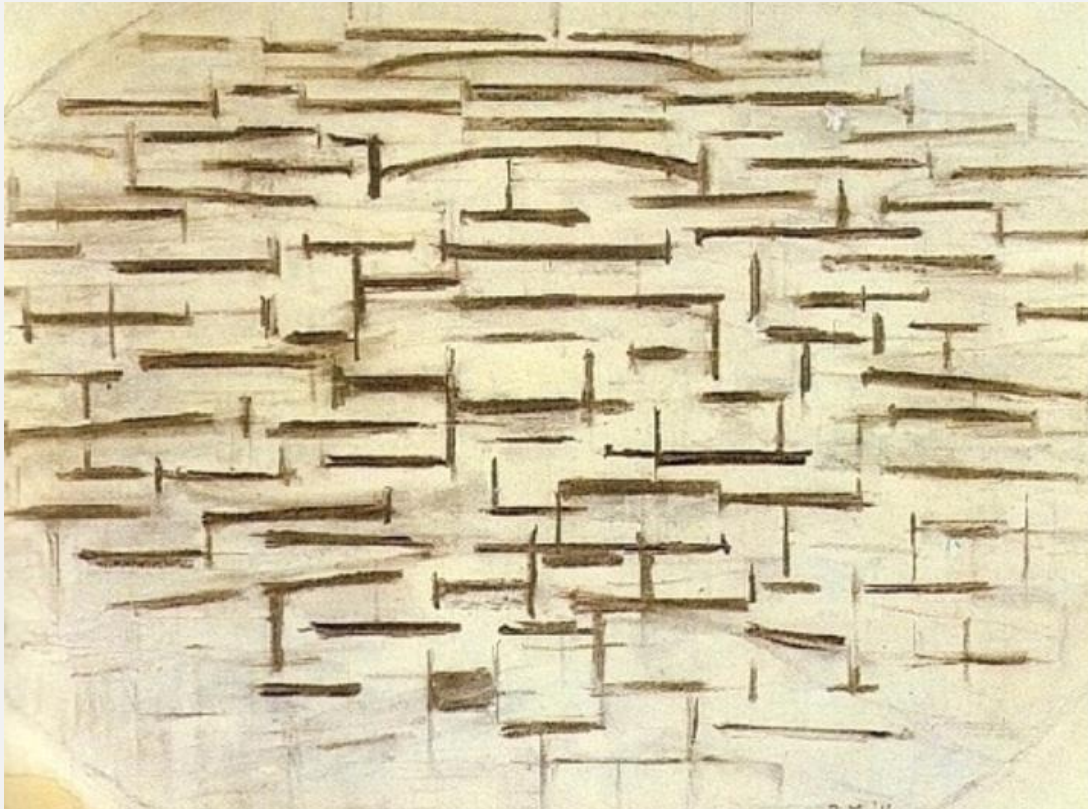
- 다음 그림은 무엇을 그린 것일까 ?



<http://blog.naver.com/PostView.nhn?blogId=jhinju&logNo=10185034539&parentCategoryNo=&categoryNo=38&viewDate=&isShowPopularPosts=true&from=search>

Programming Paradigm

- 다음 그림은 무엇을 그린 것일까 ?



Programming Paradigm

- 다음 그림은 무엇을 그린 것일까 ?



<http://blog.naver.com/PostView.nhn?blogId=jhinju&logNo=10185034539&parentCategoryNo=&categoryNo=38&viewDate=&isShowPopularPosts=true&from=search>

Programming Paradigm

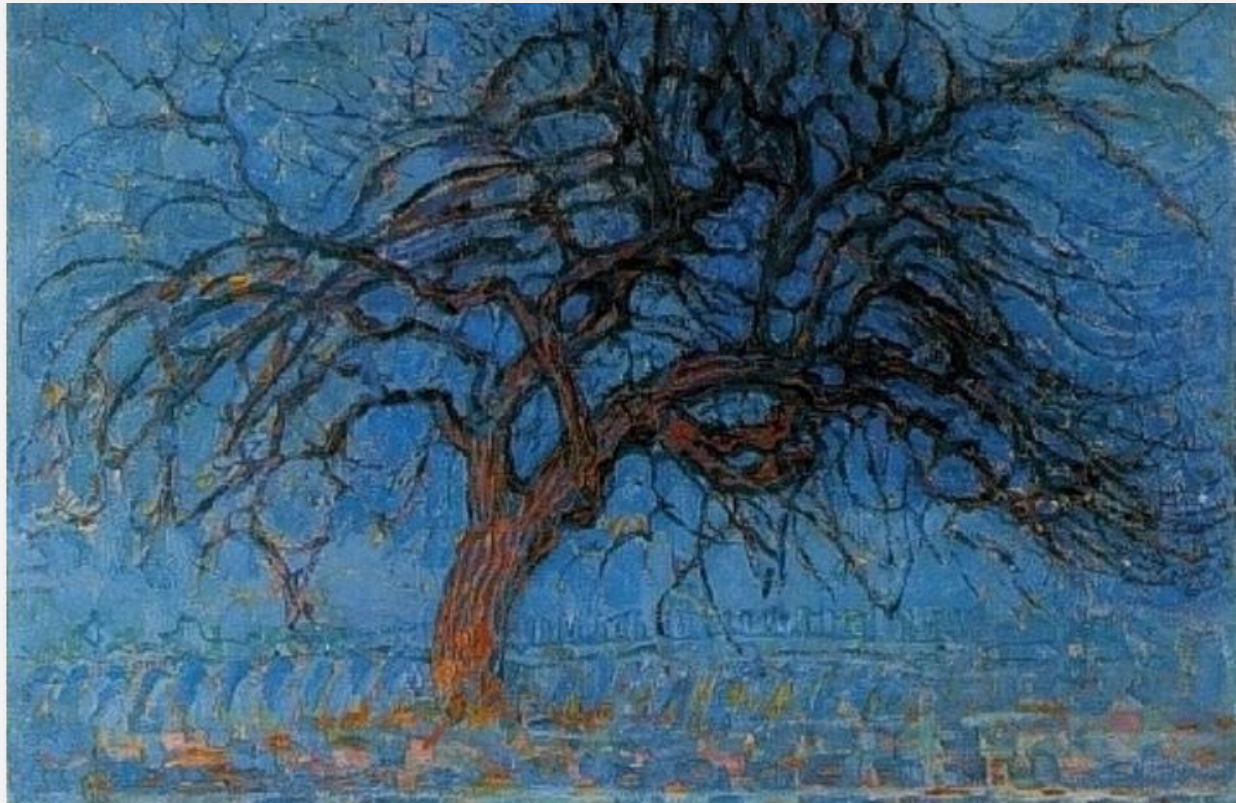
- 다음 그림은 무엇을 그린 것일까 ?



<http://blog.naver.com/PostView.nhn?blogId=jhinju&logNo=10185034539&parentCategoryNo=&categoryNo=38&viewDate=&isShowPopularPosts=true&from=search>

Programming Paradigm

- 다음 그림은 무엇을 그린 것일까 ?



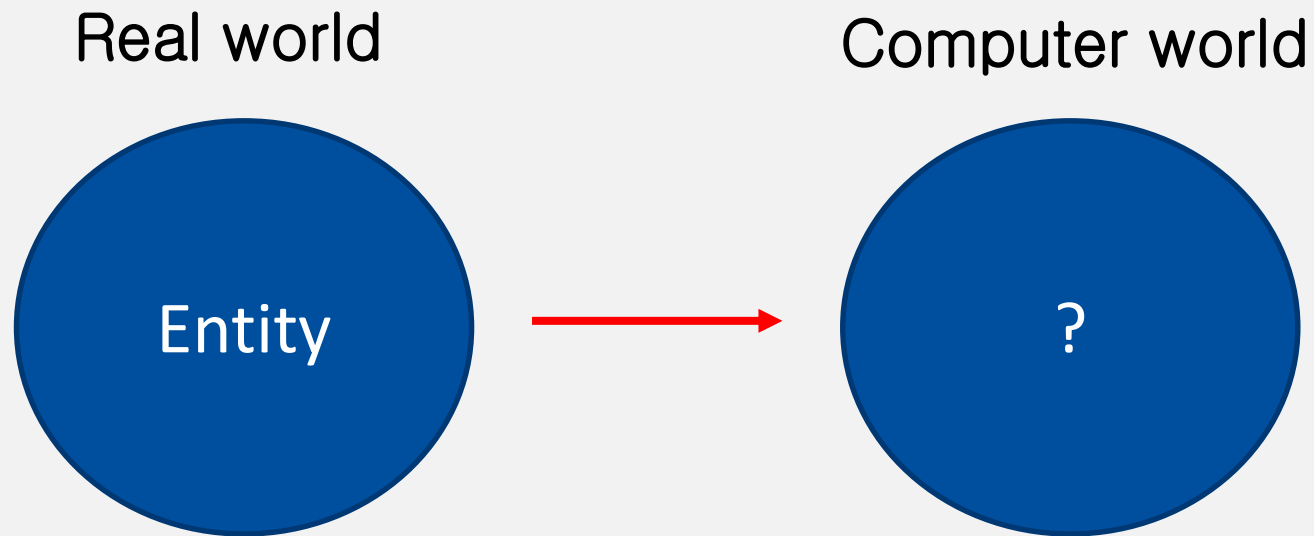
<http://blog.naver.com/PostView.nhn?blogId=jhinju&logNo=10185034539&parentCategoryNo=&categoryNo=38&viewDate=&isShowPopularPosts=true&from=search>

Programming Paradigm

- 추상화 (抽象畵) - abstract painting
- 추상화 (抽象化) – abstraction

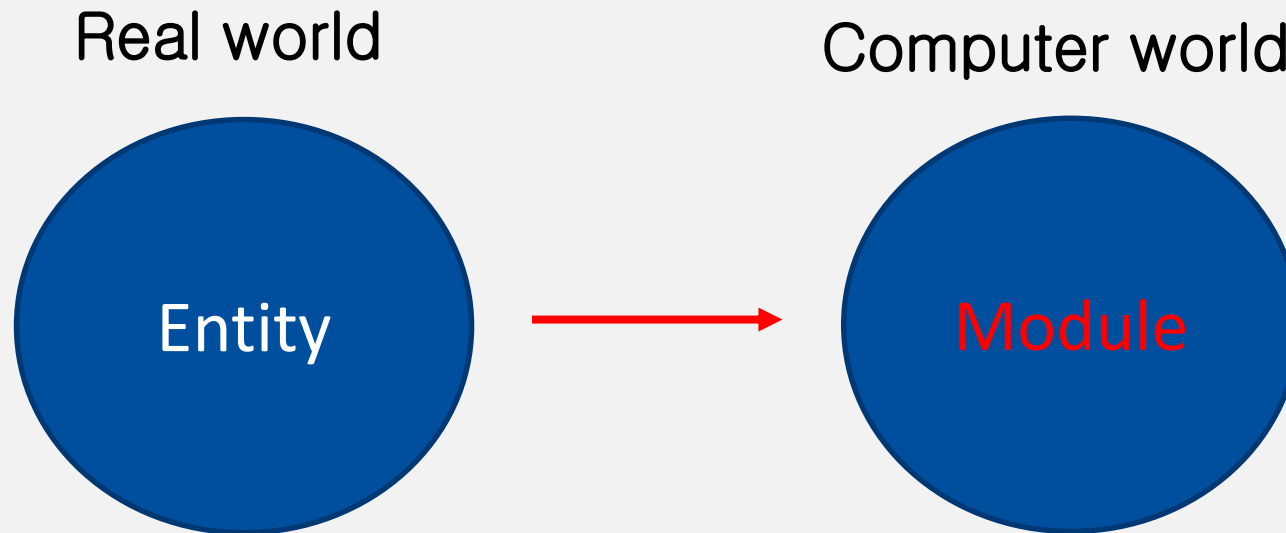
Programming Paradigm

- 모델링(Modeling) / 추상화 (抽象化, abstraction)



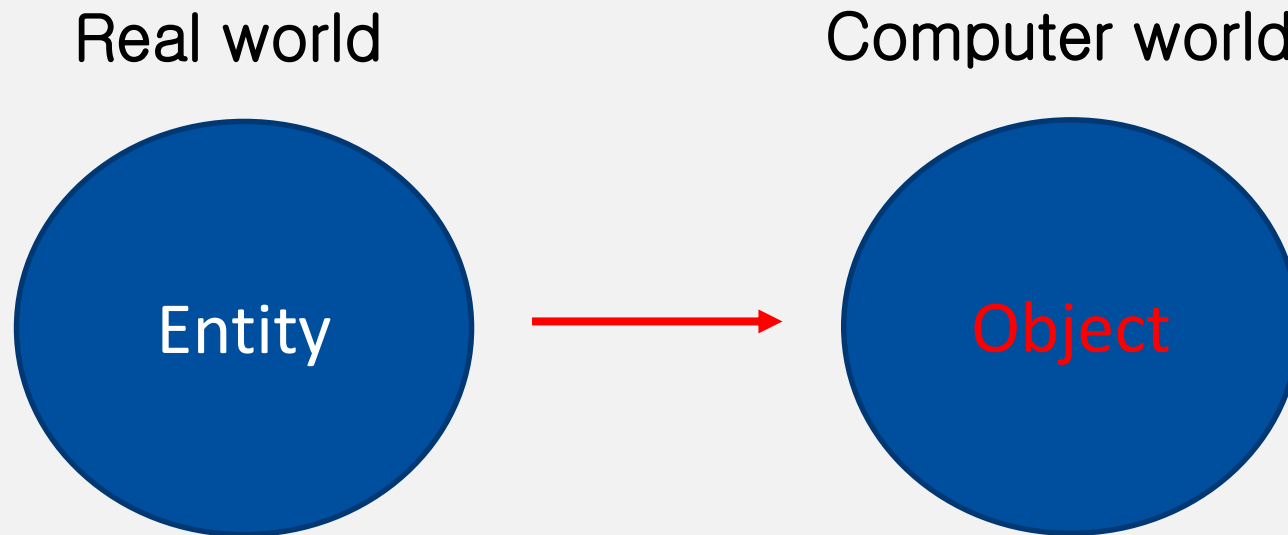
Programming Paradigm

- **Structure Programming (SP)**
 - Module
 - Structure 기반
 - Readability (가독성 : 可讀性)



Programming Paradigm

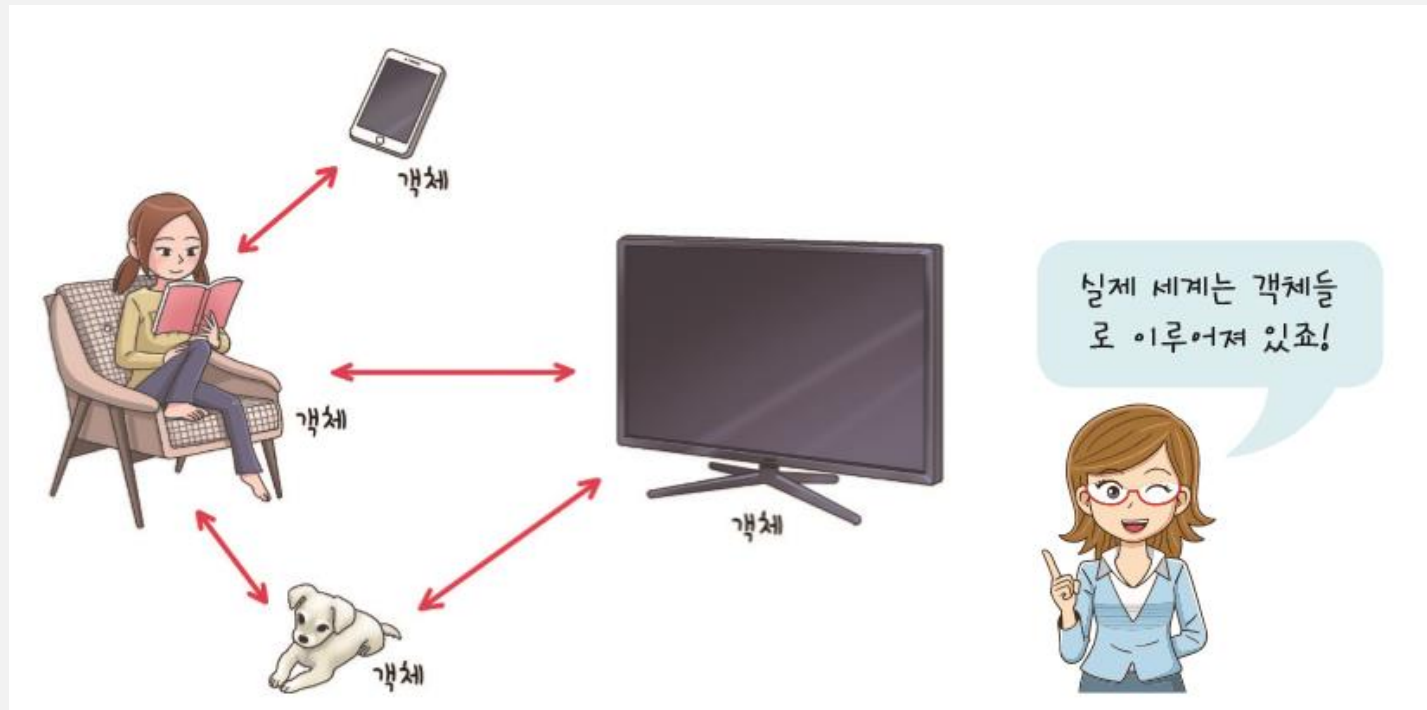
- **Object-Oriented Programming (OOP)**
 - Object
 - Encapsulation (from information hiding)
 - Reusability (재사용성 : 再使用性)



객체지향 프로그래밍

- **Object-Oriented Programming (OOP)**

우리가 살고 있는 실제 세계가 객체(Object)들로 구성되어 있는 것과 비슷하게, 소프트웨어도 객체로 구성하는 방법

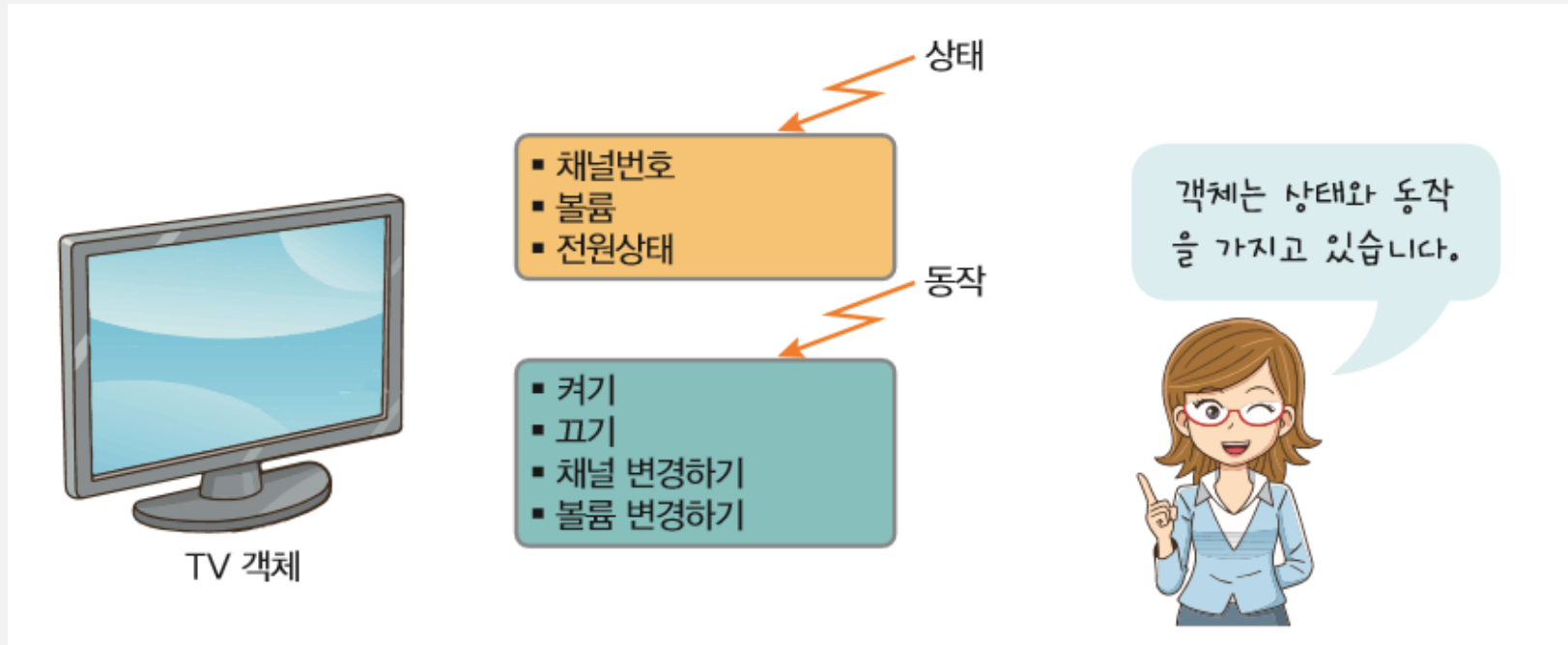


객체 (Object)

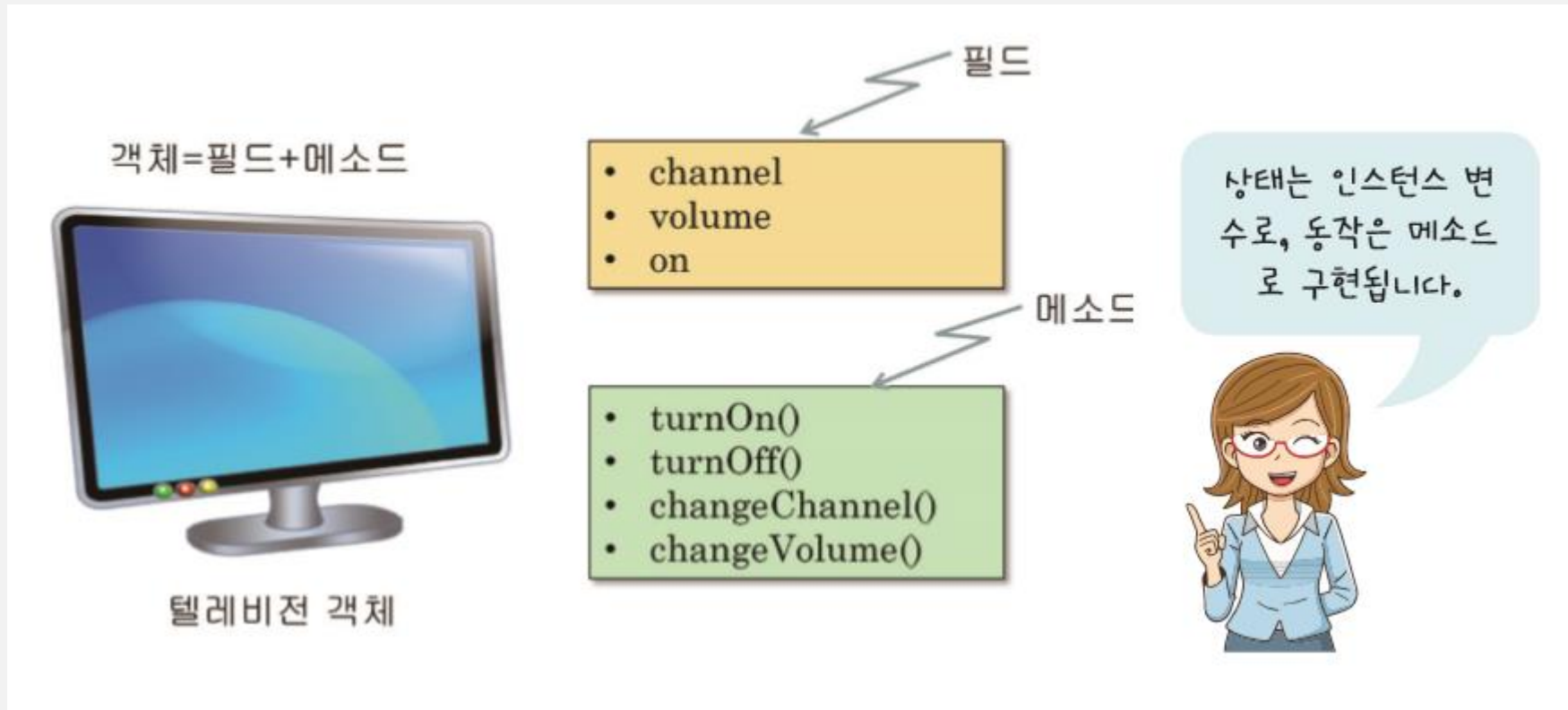
- 객체(Object) : 어떤 대상에 대해 설명 가능한 속성과 그 대상을 가지고 할 수 있는 행동을 모아놓은 것.
 - Object = attribute + method
 - Object의 다른 이름 : 인스턴스(instance)
- Attribute (member variable)
 - 대상에 대한 특징 혹은 대상에 대해 알고 있는 정보
 - 다루고자 하는 Data (or field)
- Method (member function)
 - 대상의 행동 또는 대상을 가지고 할 수 있는 일
 - Data를 manipulation할 수 있는 operation (function)

객체 (Object)

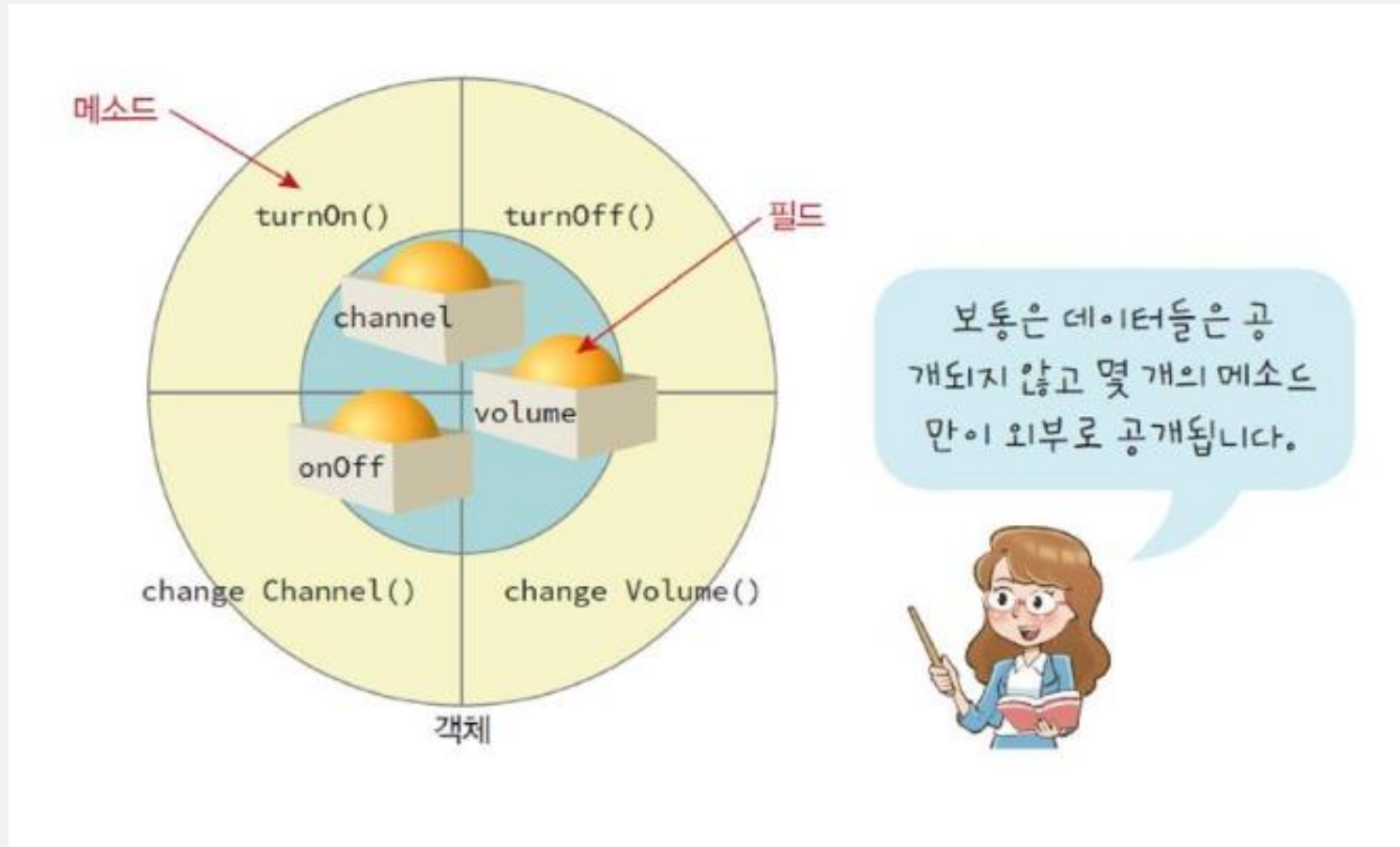
- 객체는 상태와 동작을 가지고 있음
 - 객체의 상태(state)는 객체의 속성(attribute)
 - 객체의 동작(behavior)은 객체가 취할 수 있는 동작(기능 : method)



인스턴스 변수(Instance Variable)와 메소드(Method)



인스턴스 변수(Instance Variable)와 메소드(Method)




<https://fehoon.tistory.com/101>

객체지향 프로그래밍

- **Object-Oriented Programming (OOP) Paradigm**

- Real world의 entity를 computer world의 object으로 mapping(abstraction)
- Basic philosophy of OOP : maximize the level of software **reusability**

- **Instantiation**

- Class  objects
instantiation

- **Inheritance**

- Polymorphism
- Override / overload
- Multiple inheritance
- Interface (?)

Class

- 객체에 대한 Template를 Class라고 함
- Class로부터 만들어지는 각각의 객체들을 그 Class의 인스턴스(instances)라고 함



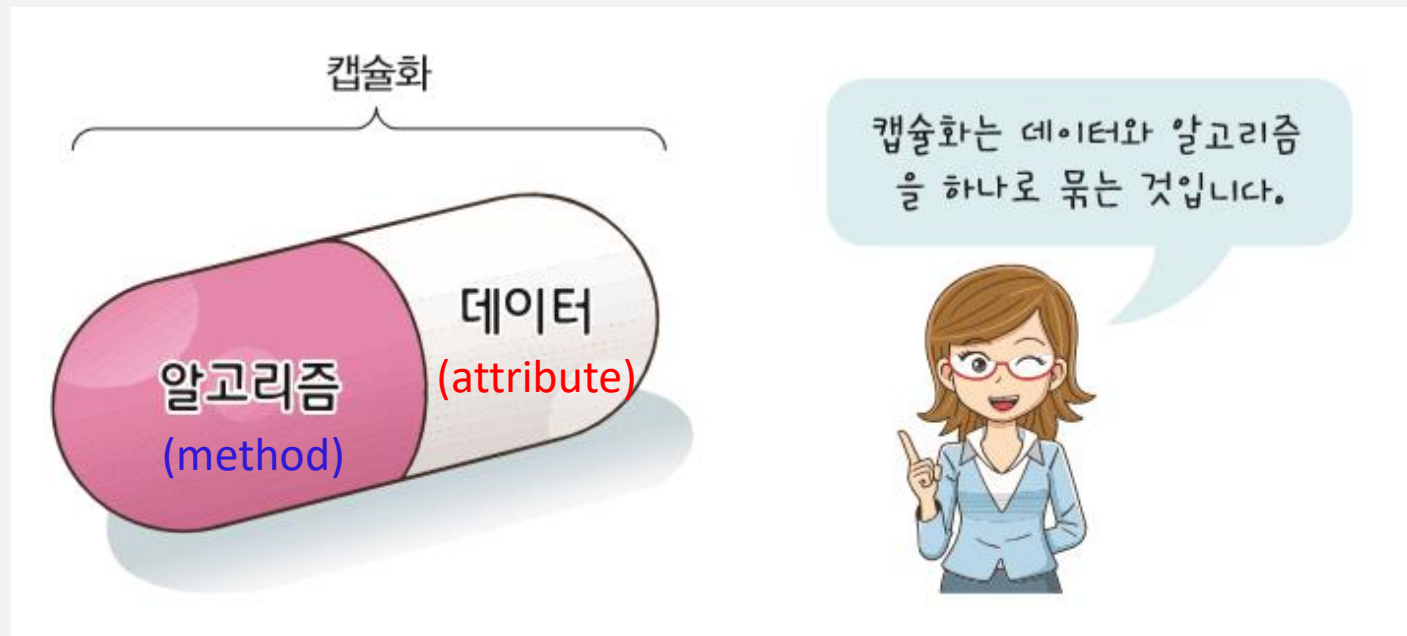
Class

- 추상적이지만 어떤 요소들에 대한 공통적이고 일반적인 '정의'를 할 수 있는 개념
 - 객체의 속성(attribute)을 저장하기 위한 변수들과
 - 객체의 행동을 수행하기 위한 method들을

같은 이름공간으로 묶은 집합체

Class

- Encapsulation(캡슐화)
 - 처리하고자 하는 data(attribute)와 이 data를 processing하는 algorithm(function, operation, method)을 하나로 묶어 공용 interfaces만 제공하고 구현 세부 사항을 감추는 것(information hiding)을 캡슐화(encapsulation)라고 함
 - Reusability(재사용성) 지원



Class

- 점(.) 표기법
 - 객체의 이름과 속성 사이, 이름과 method 사이에 점(.)을 이용
 - Python 표기법으로 점 표기법(dot notation)이라고 함

Class

- Class는 크게 Class 정의(id), member variable, member method 등 3 가지 부분으로 구성
- Member variable과 method는 반드시 필요한 것은 아님

```
class Ball:                                ← Class 정의 (id)

    size = 0
    direction = "default"                  ← Member Variable

    def bounce(self):
        if self.direction == "down":
            self.direction = "up"          ← Member method
```

self는 Python 만의 독특한 변수로 class 내에서 정의되는 함수는 무조건 첫번째 인자로 self를 사용하여야 함 → 후에 설명

Class

- 간단한 Ball class

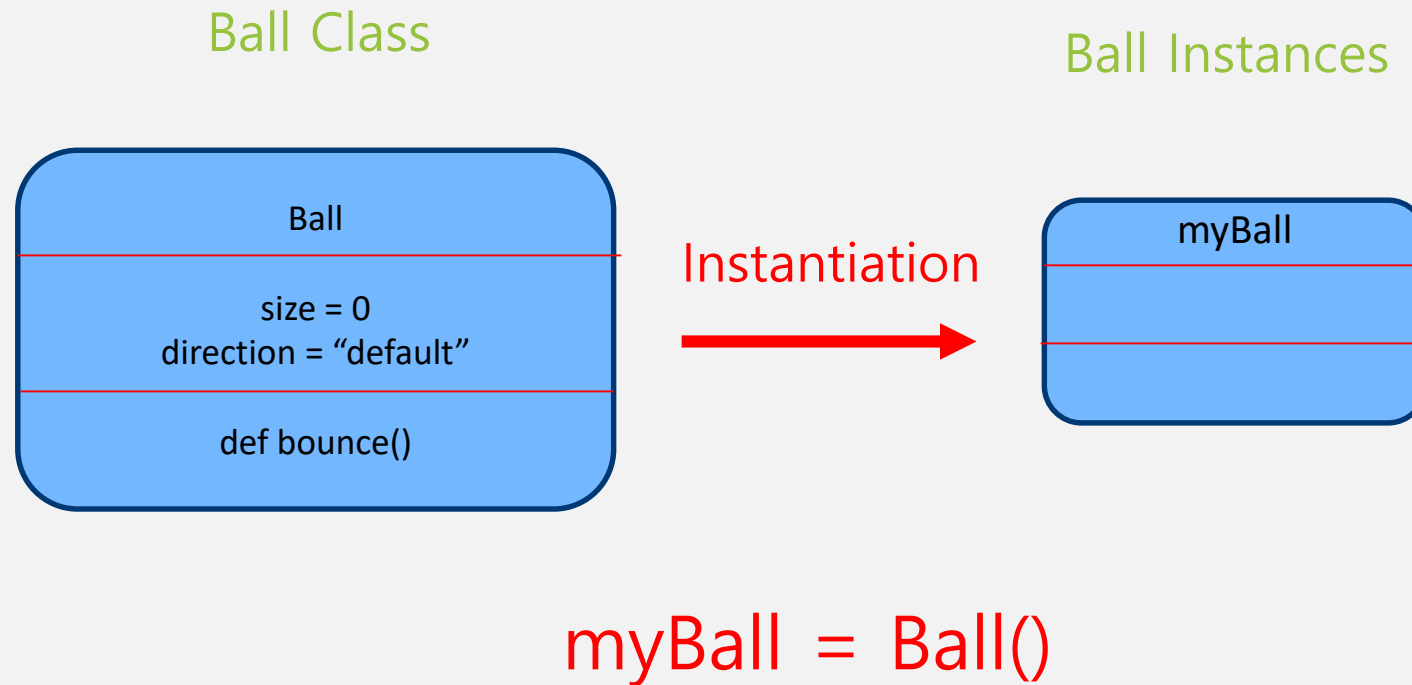
```
Ball.py
File Edit Format Run Options Window Help
1 # create a simple Ball class
2
3 class Ball:                                # Class ID
4
5     size = ArithmeticError                 # Member variable
6     direction = "default"
7
8     def bounce(self):                       # Member function (method)
9         if self.direction == "down":
10             self.direction = "up"
11
```

Instance (Object)

- Class를 사용하여 만든 실제 객체
- Class는 instance를 만드는 하나의 틀(template)
- Instance를 통해 변수나 함수의 이름을 찾는 순서
 - Instance 영역 -> Class 영역 -> Global 영역



Instance(Object) 생성



Instance 생성

- 기본적으로 instance는 생성이 완료된 직후 원본 class와 동일한 데이터와 함수를 가짐.

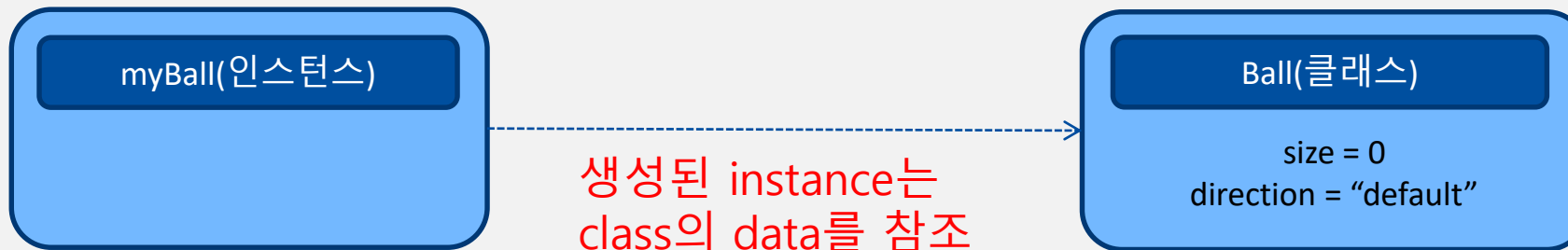
```
class Ball:
    size = 0
    direction = "default"

    def bounce(self):
        if self.direction == "down":
            self.direction = "up"
```

간단한 Ball class 생성

```
myBall = Ball()
```

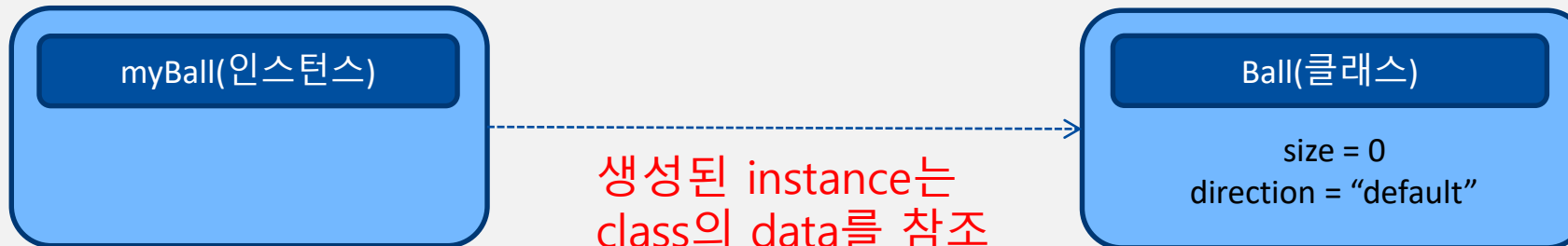
Class의 instance 생성



Instance 생성

```
Ball-2.py
File Edit Format Run Options Window Help
1 # create a simple Ball class
2
3 class Ball:
4
5     size = 0
6     direction = "default"
7
8     def bounce(self):                # This is a method
9         if self.direction == "down": #
10             self.direction = "up"    #
11
12 # main
13 myBall = Ball()
14
15 print("1) myBall.size = ", myBall.size)
16 print("2) myBall.direction = ", myBall.direction)
17
```

```
1) myBall.size = 0
2) myBall.direction = default
```



Instance attribute 값 변경

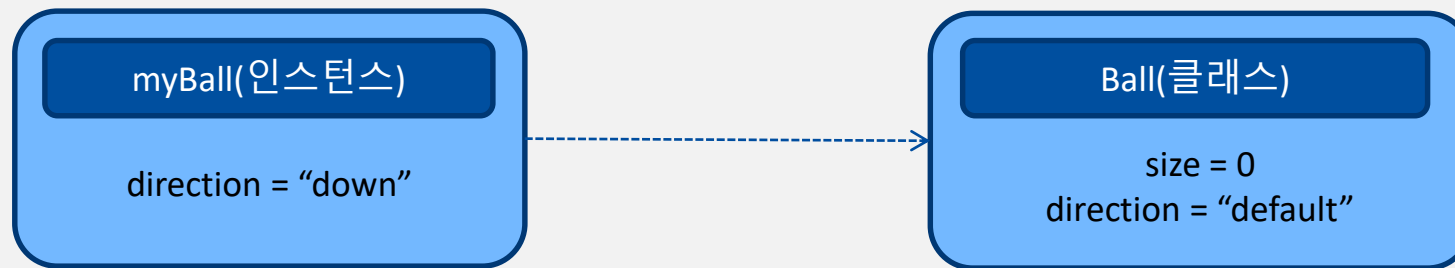
```
class Ball:
    size = 0
    direction = "default"

    def bounce(self):
        if self.direction == "down":
            self.direction = "up"
```

```
myBall = Ball()
```

```
myBall.direction = "down"
```

← Instance의 멤버 변수 값 변경

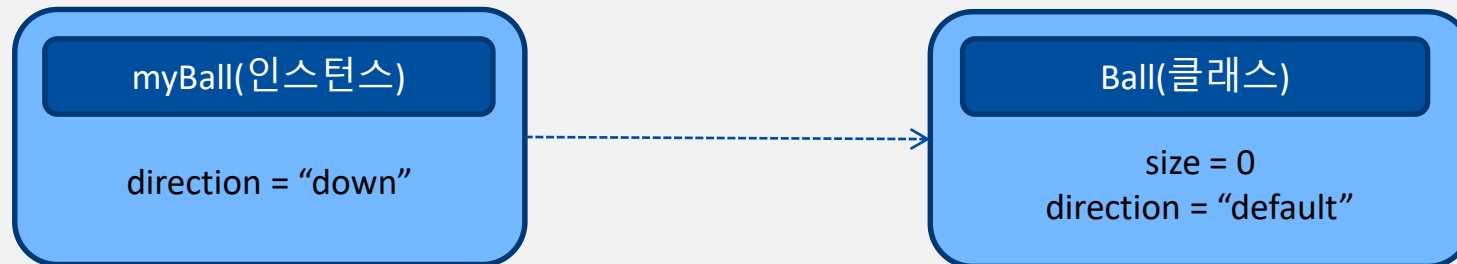


Instance에 특화된 data는 instance의 이름공간에 저장

Instance attribute 값 변경

```
Ball-2-1.py
File Edit Format Run Options Window Help
1 # create a simple Ball class
2
3 class Ball:
4     size = 0
5     direction = "default"
6
7     def bounce(self):          # This is a method
8         if self.direction == "down": #
9             self.direction = "up"    #
10
11
12 # main
13 myBall = Ball()
14 myBall.direction = "down"
15
16 print("1) myBall.size = ", myBall.size)
17 print("2) myBall.direction = ", myBall.direction)
18
```

```
1) myBall.size = 0
2) myBall.direction = down
```



Instance에 특화된 data는 instance의 이름공간에 저장

Instance attribute 변경 및 method 호출

```
class Ball:
    size = 0
    direction = "default"

    def bounce(self):
        if self.direction == "down":
            self.direction = "up"
```

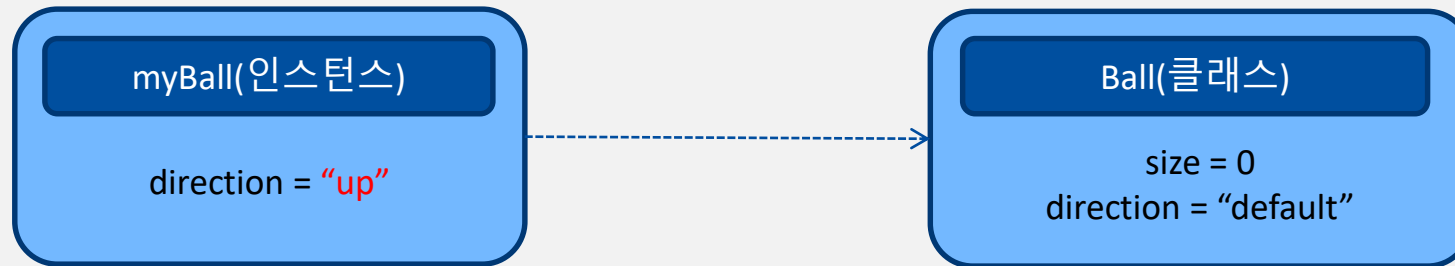
```
myBall = Ball()
```

```
myBall.direction = "down"
```

```
myBall.bounce()
```

← Instance의 멤버 변수 값 변경

← Method 호출



Instance에 특화된 data는 instance의 이름공간에 저장

```

1 # create a simple Ball class
2
3 class Ball:
4     size = 0
5     direction = "default"
6
7     def bounce(self):
8         if self.direction == "down":
9             self.direction = "up"
10
11 # main
12 myBall = Ball()
13 myBall.direction = "down"
14
15 print("1) myBall.size = ", myBall.size)
16 print("2) myBall.direction = ", myBall.direction)
17
18 myBall.bounce()
19 print("\n3) myBall.size = ", myBall.size)
20 print("4) myBall.direction = ", myBall.direction)
21
22

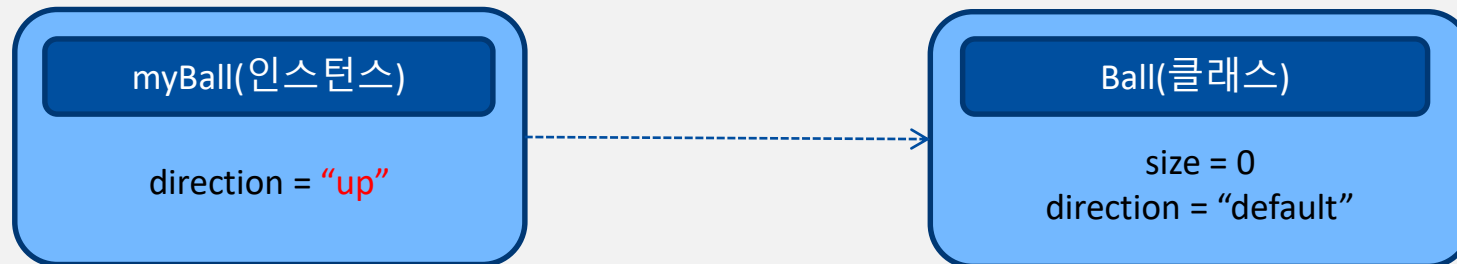
```

```

1) myBall.size = 0
2) myBall.direction = down

3) myBall.size = 0
4) myBall.direction = up

```



Instance에 특화된 data는 instance의 이름공간에 저장

Instance 멤버 변수 동적 추가

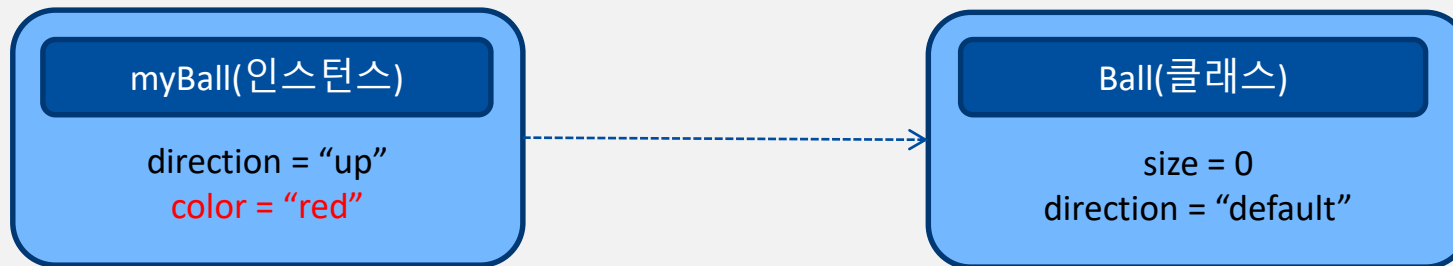
- Class와 Instance에 동적으로 멤버 변수 추가/삭제 가능.

```
class Ball:
    size = 0
    direction = "default"

    def bounce(self):
        if self.direction == "down":
            self.direction = "up"
```

```
myBall = Ball()
myBall.direction = "down"
myBall.color = "red"
```

myBall 객체에만 color 멤버
변수를 추가

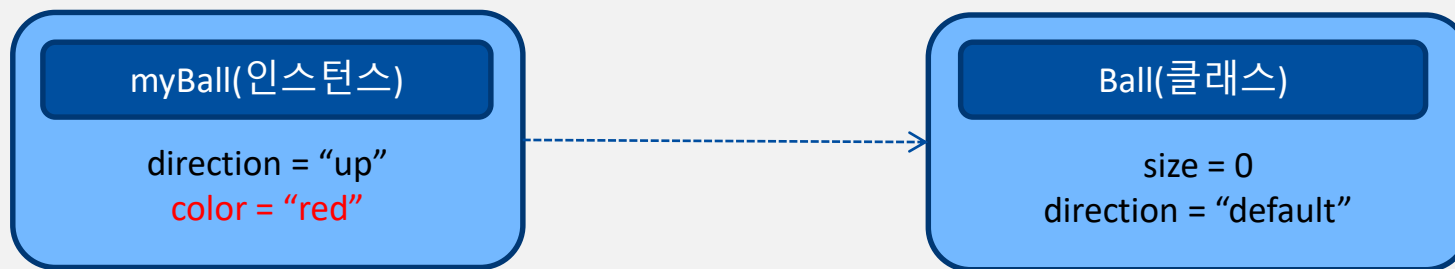


Instance에 동적으로 추가된 변수 color

```
1 # create a simple Ball class
2
3 class Ball:
4
5     size = 0
6     direction = "default"
7
8     def bounce(self):          # This is a method
9         if self.direction == "down": #
10             self.direction = "up"    #
11
12 # main
13 myBall = Ball()
14 myBall.direction = "down"
15 myBall.color = "red"
16
17 print("1) myBall.size = ", myBall.size)
18 print("2) myBall.direction = ", myBall.direction)
19 print("3) myBall.color = ", myBall.color)
20
21 myBall.bounce()
22 print("4) myBall.size = ", myBall.size)
23 print("5) myBall.direction = ", myBall.direction)
24
```

1) myBall.size = 0
2) myBall.direction = down
3) myBall.color = red

4) myBall.size = 0
5) myBall.direction = up



Instance에 동적으로 추가된 변수 color

Instance를 통한 Class 참조

- 인스턴스가 자신을 생성한 Class를 참조하기 위해 instance의 내장 속성 '`__class__`'를 사용

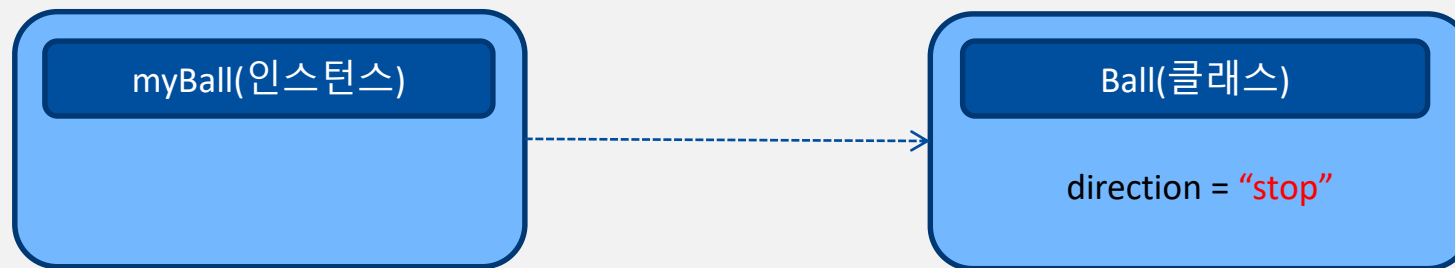
```
class Ball:
    direction = "default"

    def bounce(self):
        if self.direction == "down":
            self.direction = "up"
```

```
myBall = Ball()
```

```
myBall.__class__.direction = "stop"
```

'`__class__`' 속성을
이용해 class 데이
터를 변경



생성된 instance를 통해 class 변수 값 변경

```

1 # create a simple Ball class
2
3 class Ball:
4     size = 0
5     direction = "default"
6
7     def bounce(self):          # This is a method
8         if self.direction == "down": #
9             self.direction = "up"    #
10
11 # main
12 myBall = Ball()
13 myBall.direction = "down"
14 myBall.color = "red"
15
16 print("1) myBall.size = ", myBall.size)
17 print("2) myBall.direction = ", myBall.direction)
18 print("3) myBall.color = ", myBall.color)
19
20 myBall.bounce()
21 print("\n4) myBall.size = ", myBall.size)
22 print("5) myBall.direction = ", myBall.direction)
23
24 print("6) Class Ball의 direction 값 = ", myBall.__class__.direction)
25
26 myBall.__class__.direction = "side"
27 print("7) Class Ball의 direction 값 = ", myBall.__class__.direction)
28
29

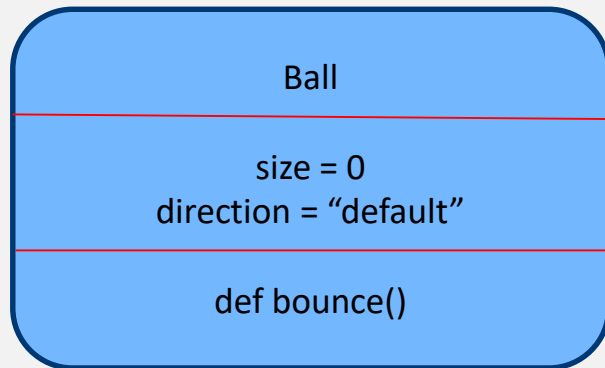
```

- 1) myBall.size = 0
- 2) myBall.direction = down
- 3) myBall.color = red
- 4) myBall.size = 0
- 5) myBall.direction = up
- 6) Class Ball의 direction 값 = default
- 7) Class Ball의 direction 값 = side

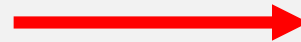


Instance 생성

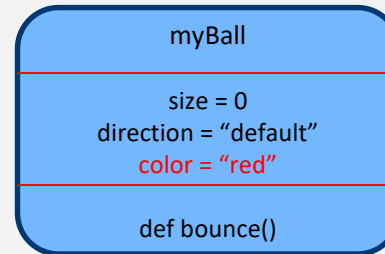
Ball Class



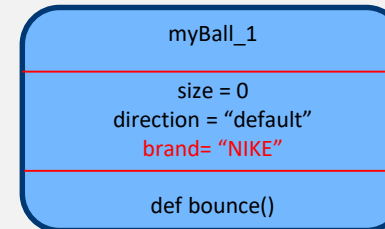
Instantiation



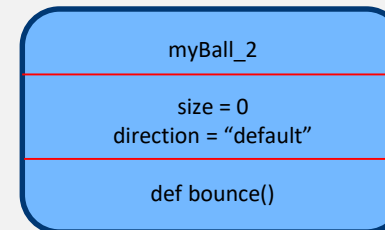
Ball Instances



Instance 1
: myBall



Instance 2
: myBall_1



Instance 3
: myBall_2

```

1 class Ball:
2
3     size = 0
4     direction = "default"
5
6     def bounce(self):
7         print("\nb-1) in bounce, self :", self)
8         if self.direction == "down":
9             self.direction = "up"
10            print("\nb-2) self.direction = ", self.direction)
11
12        def change_size(self, value):
13            print("\nc-1) value = {}, self = {}".format(value, self))
14            print("\nc-2) self.size = ", self.size)
15
16            self.size = self.size + value
17            print("\nc-3) self.size = ", self.size)
18
19 # myBall 객체 생성
20 print ("\n1) myBall 객체 생성")
21 myBall = Ball()
22
23 myBall.direction = "down"
24 myBall.size = 10
25 myBall.color = "red"
26 print ("\n2) myBall.direction = ", myBall.direction)
27 print ("\n    myBall.size = ", myBall.size)
28 print ("\n    myBall.color = ", myBall.color)
29
30 myBall.bounce()
31 print ("\nb-3) myBall.direction = ", myBall.direction)
32
33 print ("\nb-4) call change_size(value)")
34 myBall.change_size(10)
35
36 print ("\nb-5) myBall.size = ", myBall.size)
37

```

```

1) myBall 객체 생성
2) myBall.direction = down
   myBall.size = 10
   myBall.color = red

```

```

b-1) in bounce, self : <__main__.Ball object at 0x0397C5B0>
b-2) self.direction = up

```

```
3) myBall.direction = up
```

```
4) call change_size(value)
```

```

c-1) value = 10, self = <__main__.Ball object at 0x0397C5B0>
c-2) self.size = 10
c-3) self.size = 20

```

```
5) myBall.size = 20
```

```

6) myBall_1 객체 생성
7) myBall_1.direction = side
   myBall_1.size = 30
   myBall_1.brand = NIKE

```

```
8) call bounce()
```

```

b-1) in bounce, self : <__main__.Ball object at 0x039DC3B8>
9) myBall_1.direction = side

```

```
10) call change_size()
```

```

c-1) value = 10, self = <__main__.Ball object at 0x039DC3B8>
c-2) self.size = 30
c-3) self.size = 40
11) myBall_1.size = 40

```

```

12) myBall_2 객체 생성
13) myBall_2.direction = left
    myBall_2.size = 50

```

```
14) call bounce()
```

```

b-1) in bounce, self : <__main__.Ball object at 0x039DC340>
15) myBall_2.direction = left
16) call change_size(value)

```

```

c-1) value = 10, self = <__main__.Ball object at 0x039DC340>
c-2) self.size = 50
c-3) self.size = 60
23) myBall_2.size = 60

```

```

38 # myBall_1 객체 생성
39 print ("Wn6) myBall_1 객체 생성")
40 myBall_1 = Ball()
41
42 myBall_1.direction = "side"
43 myBall_1.size = 30
44 myBall_1.brand = "NIKE"
45
46 print ("7) myBall_1.direction = ", myBall_1.direction)
47 print ("    myBall_1.size = ", myBall_1.size)
48 print ("    myBall_1.brand = ", myBall_1.brand)
49
50 print ("Wn8) call bounce()")
51 myBall_1.bounce()
52 print ("9) myBall_1.direction = ", myBall_1.direction)
53
54 print ("Wn10) call change_size()")
55 myBall_1.change_size(10)
56 print ("11) myBall_1.size = ", myBall_1.size)
57
58 # myBall_2 객체 생성
59 print ("Wn12) myBall_2 객체 생성")
60 myBall_2 = Ball()
61
62 myBall_2.direction = "left"
63 myBall_2.size = 50
64
65 print ("13) myBall_2.direction = ", myBall_2.direction)
66 print ("    myBall_2.size = ", myBall_2.size)
67
68 print ("Wn14) call bounce()")
69 myBall_2.bounce()
70 print ("15) myBall_2.direction = ", myBall_2.direction)
71
72 print ("16) call change_size(value)")
73 myBall_2.change_size(10)
74 print ("23) myBall_2.size = ", myBall_2.size)

```

```

1) myBall 객체 생성
2) myBall.direction = down
   myBall.size = 10
   myBall.color = red

b-1) in bounce, self : <__main__.Ball object at 0x0397C5B0>
b-2) self.direction = up

3) myBall.direction = up

4) call change_size(value)

c-1) value = 10, self = <__main__.Ball object at 0x0397C5B0>
c-2) self.size = 10
c-3) self.size = 20

5) myBall.size = 20

6) myBall_1 객체 생성
7) myBall_1.direction = side
   myBall_1.size = 30
   myBall_1.brand = NIKE

8) call bounce()

b-1) in bounce, self : <__main__.Ball object at 0x039DC3B8>
9) myBall_1.direction = side

10) call change_size()

c-1) value = 10, self = <__main__.Ball object at 0x039DC3B8>
c-2) self.size = 30
c-3) self.size = 40
11) myBall_1.size = 40

12) myBall_2 객체 생성
13) myBall_2.direction = left
    myBall_2.size = 50

14) call bounce()

b-1) in bounce, self : <__main__.Ball object at 0x039DC340>
15) myBall_2.direction = left
16) call change_size(value)

c-1) value = 10, self = <__main__.Ball object at 0x039DC340>
c-2) self.size = 50
c-3) self.size = 60
23) myBall_2.size = 60

```


Class 형식

전체적인 구조



```
class 클래스 이름 :
```

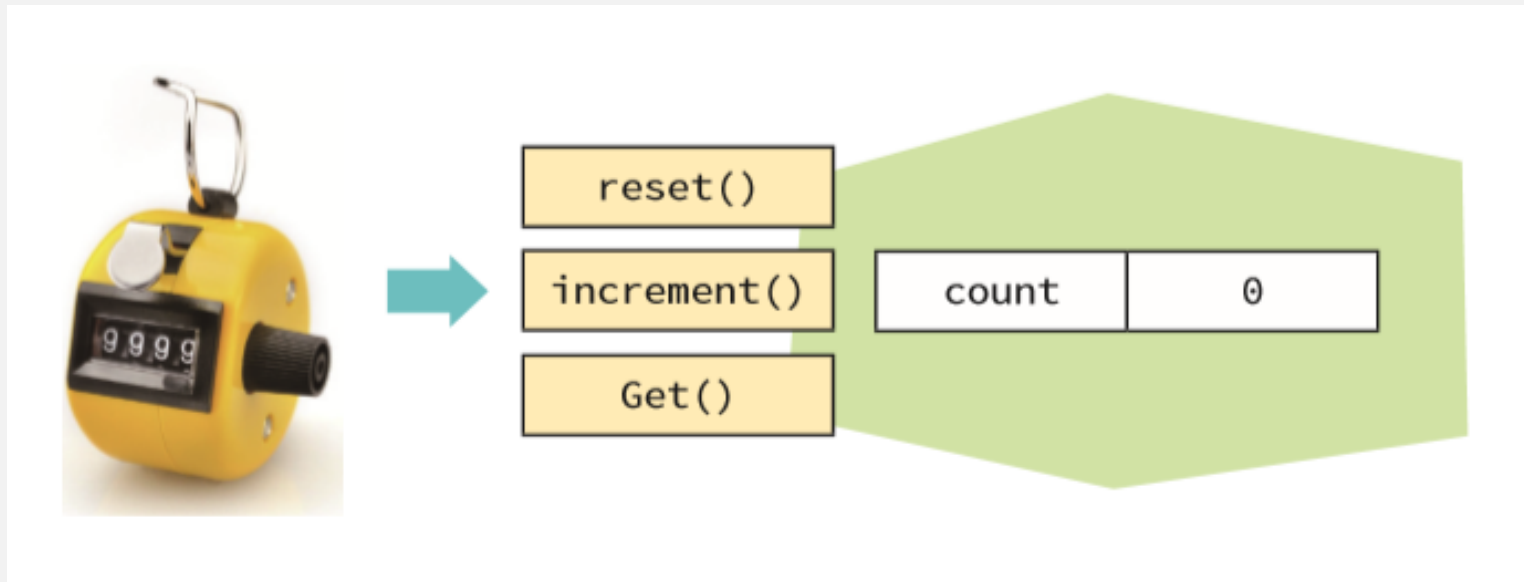
```
def 메소드1 (self, ...):  
    ...
```

```
def 메소드2 (self, ...):  
    ...
```

메소드를 정의한다.

Class의 예

- Counter Class를 작성하여 보자. Counter Class는 기계식 계수기를 나타내며 경기장이나 콘서트에 입장하는 관객 수를 세기 위하여 사용할 수 있음



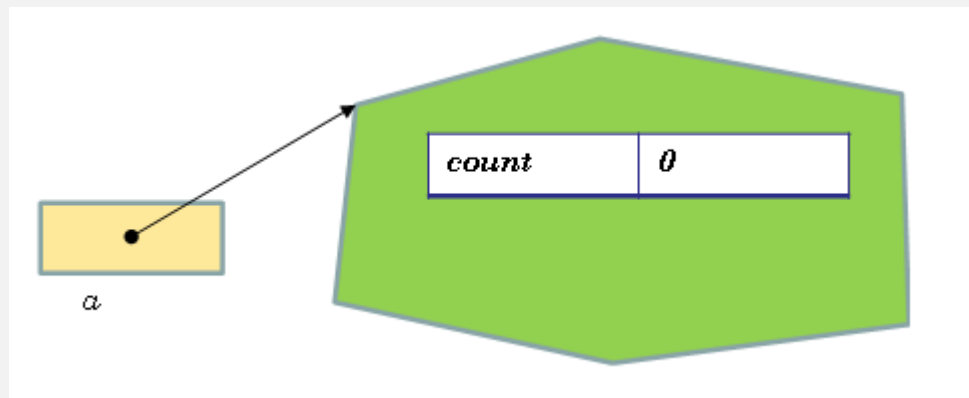
Counter Class

```
class Counter:  
    def reset(self):  
        self.count = 0  
    def increment(self):  
        self.count += 1  
    def get(self):  
        return self.count
```

객체 생성

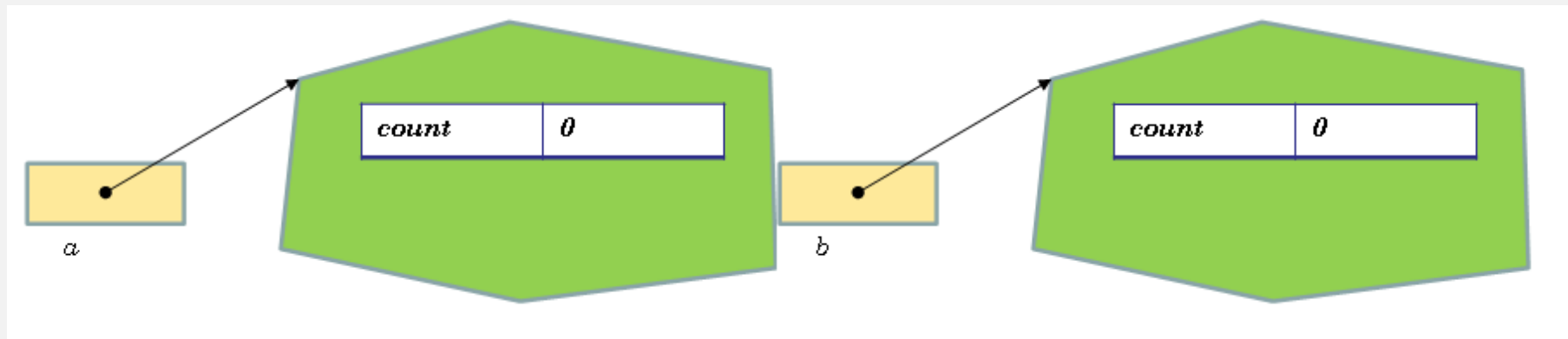
```
a = Counter()  
a.reset()  
a.increment()  
print("카운터 a의 값은", a.get())
```

카운터 a의 값은 1



객체 2개 생성하기

```
a = Counter()  
b = Counter()  
  
a.reset()  
b.reset()
```



```

1 class Counter:
2
3     count = 0
4
5     def reset(self):
6         self.count = 0
7         print("reset) self.count = {}".format(self.count))
8
9     def increment(self):
10        self.count += 1
11        print("increment) self.count = {}".format(self.count))
12
13    def get(self):
14        print("get) self.count = {}".format(self.count))
15        return self.count
16
17 # main
18 a = Counter()
19 b = Counter()
20
21 a.reset()
22 print("\n1) 카운터 a의 값 :", a.get())
23
24 a.increment()
25 print("\n2) 카운터 a의 값 :", a.get())
26
27 b.reset()
28 print("\n3) 카운터 b의 값 :", b.get())
29
30 b.increment()
31 b.increment()
32 b.increment()
33 print("\n4) 카운터 b의 값 :", b.get())
34
35 print("\n5) 카운터 a의 값 :", a.get())

```

```
reset) self.count = 0
```

```
get) self.count = 0
```

1) 카운터 a의 값 : 0

```
increment) self.count = 1
```

```
get) self.count = 1
```

2) 카운터 a의 값 : 1

```
reset) self.count = 0
```

```
get) self.count = 0
```

3) 카운터 b의 값 : 0

```
increment) self.count = 1
```

```
increment) self.count = 2
```

```
increment) self.count = 3
```

```
get) self.count = 3
```

4) 카운터 b의 값 : 3

```
get) self.count = 1
```

5) 카운터 a의 값 : 1

객체 초기화 - 생성자(constructor) method

- Class로부터 객체 생성(instantiation)시 자동으로 실행되는 특수 method
- 객체(또는 instance)가 생성(instantiation)될 때 자동으로 호출
- 특수 method '`__init__()`'으로 정의.
 - 특수 method : Class 생성 시 Python에서 (자동으로) 만들어주는 method (user가 overload 할 수 있음) – p.50

```
class Ball:  
    def __init__(self, color, size, direction):  
        self.color = color  
        self.size = size  
        self.direction = direction
```

`__init__()`
method 정의

객체 초기화 - 생성자(constructor) method

- 객체를 생성하면서 속성값을 설정하기 위한 방법.
- Constructor argument

생성자를 통해 인스턴스 생성 시 초기화 할 멤버 변수 값을 인자로 전달

```
class Ball:
    def __init__(self, color, size, direction):
        self.color = color
        self.size = size
        self.direction = direction
```

```
myBall = Ball("red", "small", "down")
```

myBall instance 생성과 동시에
myBall 멤버 변수 color="red",
size="small", direction = "down"
으로 초기화

```
Constructor.py
File Edit Format Run Options Window Help
# Adding an __init__() method

class Ball:
    def __init__(self, color, size, direction):          # 생성자
        print ("c-1) 생성자 입니다.")
        self.color = color                             # __init__() method
        self.size = size
        self.direction = direction

        print ("c-2) self.color = ",color)
        print ("c-3) self.size = ",size)
        print ("c-4) self.direction = ",direction)

    def bounce(self):
        print ("b-1) in bounce()")
        if self.direction == "down":
            self.direction = "up"

#main
print ("0) main 시작입니다")
myBall = Ball("red", "small", "down")                  # Create an instance
                                                         # with some attributes

print ("1) I just created a myBall instance.")
print ("2) myball's size :", myBall.size)
print ("3) myball's color :", myBall.color)
print ("4) myball's direction :", myBall.direction)
print ("5) Now I'm going to bounce the ball")
print
myBall.bounce()
print ("6) Now the ball's direction : ", myBall.direction)
```

0) main 시작입니다
c-1) 생성자 입니다.
c-2) self.color = red
c-3) self.size = small
c-4) self.direction = down
1) I just created a myBall instance.
2) myball's size : small
3) myball's color : red
4) myball's direction : down
5) Now I'm going to bounce the ball
b-1) in bounce()
6) Now the ball's direction : up
>>>

생성자의 예

전체적인 구조



```
class 클래스 이름 :
```

```
    def __init__(self, ...):
```

```
        ...
```

__init__() 메소드가 생성자이다.
여기서 객체의 초기화를 담당한다.

```
class Counter:
    def __init__(self) :
        self.count = 0
    def reset(self) :
        self.count = 0
    def increment(self):
        self.count += 1
    def get(self):
        return self.count
```

```

1 class Counter:
2
3     def __init__(self):
4         print("i-1) In init, self =", self)
5         self.count = 0
6         print("i-2) self.count =", self.count, "\n")
7
8     def reset(self):
9         print("r-1) self =", self)
10        self.count = 0
11        print("r-2) self.count =", self.count, "\n")
12
13    def increment(self):
14        print("inc-1) self =", self)
15        self.count += 1
16        print("inc-2) self.count =", self.count, "\n")
17
18    def get(self):
19        print("g-1) self =", self)
20        print("g-2) self.count =", self.count, "\n")
21        return self.count
22
23    print("\n1) main start \n")
24    print("2) Create a.")
25    a = Counter()
26
27    print("3) Create b.")
28    b = Counter()
29
30    a.reset()
31    print("4) 카운터 a의 값은", a.get(), "\n")
32
33    a.increment()
34    print("5) 카운터 a의 값은", a.get())
35
36    b.reset()
37    print("6) 카운터 b의 값은", b.get())
38
39    b.increment()
40    b.increment()
41    print("7) 카운터 b의 값은", b.get())
42

```

1) main start

2) Create a.

i-1) In init, self = <__main__.Counter object at 0x0343C3A0>
i-2) self.count = 0

3) Create b.

i-1) In init, self = <__main__.Counter object at 0x0343C358>
i-2) self.count = 0

r-1) self = <__main__.Counter object at 0x0343C3A0>

r-2) self.count = 0

g-1) self = <__main__.Counter object at 0x0343C3A0>

g-2) self.count = 0

4) 카운터 a의 값은 0

inc-1) self = <__main__.Counter object at 0x0343C3A0>

inc-2) self.count = 1

g-1) self = <__main__.Counter object at 0x0343C3A0>

g-2) self.count = 1

5) 카운터 a의 값은 1

r-1) self = <__main__.Counter object at 0x0343C358>

r-2) self.count = 0

g-1) self = <__main__.Counter object at 0x0343C358>

g-2) self.count = 0

6) 카운터 b의 값은 0

inc-1) self = <__main__.Counter object at 0x0343C358>

inc-2) self.count = 1

inc-1) self = <__main__.Counter object at 0x0343C358>

inc-2) self.count = 2

g-1) self = <__main__.Counter object at 0x0343C358>

g-2) self.count = 2

7) 카운터 b의 값은 2

객체 초기화 - 생성자 method (constructor method)

- Default constructor (묵시적 생성자)
 - 생성자를 두지않고도 class를 정의 할 수가 있는데, 이를 묵시적 생성자라고 함
 - 생성자를 지정해주지 않았을 때, 묵시적으로 생성되는 기본 생성자
 - 필요에 따라 이를 **override / overload** 하여 사용
 - 이미 앞의 예제들에서 생성자를 overload하여 사용하였음

```
def __init__(self):  
    pass
```

소멸자 메소드 (destructor method)

- Instance의 reference counter가 0(zero) 이 될 때 자동으로 호출
- 메모리 해체 등의 종료작업을 위함
- 특수 method '`__del__()`'로 정의

```
class Ball:
    def __init__(self, color, size, direction):
        self.color = color
        self.size = size
        self.direction = direction
```

```
def __del__(self):
    print("Class is deleted!")
```

`__del__()` method 정의

소멸자 메소드 (destructor method)

- <https://docs.python.org/3/library/sys.html>
- `sys.getrefcount(object)`
 - Return the **reference count** of the object
 - The count returned is generally **one higher than you might expect**, because it includes the (temporary) reference as an argument to `getrefcount()`

```

1 # Adding an __del__() method
2
3 import sys
4
5 class Ball:
6
7     #member 변수
8     color = "없음"
9     size = 0
10    direction = "up"
11
12    def __init__(self, color, size, direction):    # 생성자(constructor)
13        print("\n1) 생성자 __init__() called.")
14
15        self.color = color
16        self.size = size
17        self.direction = direction
18        print("i-2) self.color = ",color)
19        print("i-3) self.size = ",size)
20        print("i-4) self.direction = ",direction)
21
22    def __del__(self):    # 소멸자(destructor)
23        print("\n1) 소멸자. Object is deleted !")
24
25    def bounce(self):
26        print("b-1) in bounce")
27        if self.direction == "down":
28            self.direction = "up"
29            print("b-1) self.direction = ", self.direction)
30
31 # main# Garbage Collection
32 # 객체가 참조될 때마다 reference count가 1 증가
33 # 참조가 해제될 때 마다 reference count가 1 감소
34 # 카운트가 0 이 되면 객체는 메모리 공간에서 삭제
35
36 print("1) main ")
37 print("\n2) myBall 객체 생성")
38 myBall = Ball("red", "small", "down")    # myBall instance 생성
39                                           # reference counter 증가 : 1
40 print("\n3) myBall의 reference count = ", sys.getrefcount(myBall))
41 print("    myBall.size = ", myBall.size)
42 print("    myBall.color = ", myBall.color)
43 print("    myBall.direction = ", myBall.direction)
44 print("4) myBall의 reference count = ", sys.getrefcount(myBall))
45

```

1) main

2) myBall 객체 생성

i-1) 생성자 __init__() called.

i-2) self.color = red

i-3) self.size = small

i-4) self.direction = down

3) myBall의 reference count = 2

myBall.size = small

myBall.color = red

myBall.direction = down

4) myBall의 reference count = 2

5) call myBall.bounce()

b-1) in bounce

b-1) self.direction = up

6) myBall.direction = up

7) myBall의 reference count = 2

8) myBall의 reference count = 3

9) myBall의 reference count = 4

10) myBall의 reference count = 3

11) myBall의 reference count = 2

12) myBall 객체에 대한 참조를 모두 해제

13) myBall의 reference count = 2

14) myBall의 size, color, direction 값들이 class member variable 초기값으로 초기화

myBall.size = 0

myBall.color = 없음

myBall.direction = up

15) myBall의 reference count = 2

d-1) 소멸자. Object is deleted !

16) 소멸자가 자동으로 불리어져서 myBall 객체가 메모리에서 사라짐.

Traceback (most recent call last):

File "C:\와소사\와소사-강의예제\Destructor.py", line 79, in <module>

print("17) myBall의 reference count = ", sys.getrefcount(myBall))

NameError: name 'myBall' is not defined


```

1) main
2) myBall 객체 생성
i-1) 생성자 __init__() called.
i-2) self.color = red
i-3) self.size = small
i-4) self.direction = down
3) myBall의 reference count = 2
   myBall.size = small
   myBall.color = red
   myBall.direction = down
4) myBall의 reference count = 2
5) call myBall.bounce()
b-1) in bounce
b-1) self.direction = up
6) myBall.direction = up
7) myBall의 reference count = 2
8) myBall의 reference count = 3
9) myBall의 reference count = 4
10) myBall의 reference count = 3
11) myBall의 reference count = 2
12) myBall 객체에 대한 참조를 모두 해제
13) myBall의 reference count = 2
14) myBall의 size, color, direction 값들이 class member variable 초기값으로 초기화
    myBall.size = 0
    myBall.color = 없음
    myBall.direction = up
15) myBall의 reference count = 2

```

```

d-1) 소멸자. Object is deleted !
16) 소멸자가 자동으로 불리어져서 myBall 객체가 메모리에서 사라짐.
Traceback (most recent call last):
  File "C:\와소사\와소사-강의예제\Destructor.py", line 78, in <module>
    print("17) myBall의 reference count = ", sys.getrefcount(myBall))
NameError: name 'myBall' is not defined

```

```

46 print("\n5) call myBall.bounce()")
47 myBall.bounce()
48 print("\n6) myBall.direction = ", myBall.direction)
49 print("\n7) myBall의 reference count = ", sys.getrefcount(myBall))
50
51 # reference count 증가 예
52 yourBall = myBall
53 print("\n8) myBall의 reference count = ", sys.getrefcount(myBall))
54
55 herBall = myBall
56 print("\n9) myBall의 reference count = ", sys.getrefcount(myBall))
57
58 del yourBall
59 print("\n10) myBall의 reference count = ", sys.getrefcount(myBall))
60
61 del herBall
62 print("\n11) myBall의 reference count = ", sys.getrefcount(myBall))
63
64 # myBall 객체에 대한 참조를 모두 해제
65 print("\n12) myBall 객체에 대한 참조를 모두 해제")
66
67 del myBall.size, myBall.color, myBall.direction
68 print("\n13) myBall의 reference count = ", sys.getrefcount(myBall))
69
70 print("\n14) myBall의 size, color, direction 값들이 class member variable 초기값으로 초기화")
71 print("    myBall.size = ", myBall.size)
72 print("    myBall.color = ", myBall.color)
73 print("    myBall.direction = ", myBall.direction)
74 print("\n15) myBall의 reference count = ", sys.getrefcount(myBall))
75
76 del myBall
77 print("\n16) 소멸자가 자동으로 불리어져서 myBall 객체가 메모리에서 사라짐.")
78 print("\n17) myBall의 reference count = ", sys.getrefcount(myBall))

```

Method 정의

- Method는 Class 안에 정의된 함수이므로 함수를 정의하는 것과 아주 유사함.
하지만 첫 번째 매개변수는 항상 `self`이어야 함.

```
class Television:
    def __init__(self, channel, volume, on):
        self.channel = channel
        self.volume = volume
        self.on = on

    def show(self):
        print(self.channel, self.volume, self.on)

    def setChannel(self, channel):
        self.channel = channel

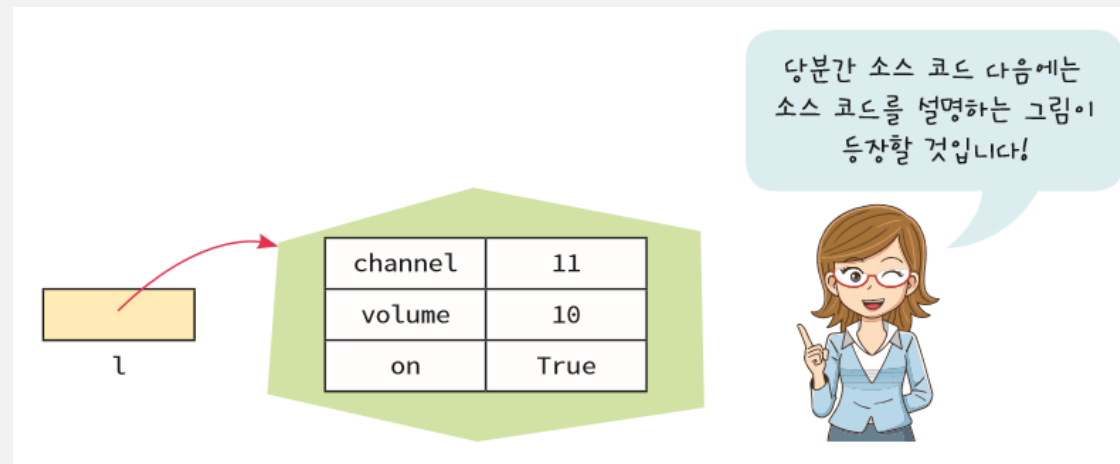
    def getChannel(self):
        return self.channel
```

Method호출

```
t = Television(9, 10, True)

t.show()
t.setChannel(11)
t.show()
```

```
9 10 True
11 10 True
```



```

class Television:
    def __init__(self, channel, volume, on):
        print("In init..")
        self.channel = channel
        self.volume = volume
        self.on = on
        print("init) self.channel =", self.channel)
        print("      self.volume =", self.volume)
        print("      self.on =", self.on)

    def show(self):
        print("show) self.channel =", self.channel)
        print("      self.volume =", self.volume)
        print("      self.on =", self.on)

    def getChannel(self):
        print("getChannel) self.channel =", self.channel)
        return self.channel

    def getVolume(self):
        print("getVolume) self.volume =", self.volume)
        return self.volume

    def getOn(self):
        print("getOn) self.On =", self.on)
        return self.on

    def setChannel(self, channel):
        self.channel = channel
        print("setChannel) self.channel =", self.channel)

    def setVolume(self, volume):
        self.volume = volume
        print("setVolume) self.volume =", self.volume)

```

1) main start

2) 객체 t 생성

In init..

init) self.channel = 9

self.volume = 10

self.on = True

show) self.channel = 9

self.volume = 10

self.on = True

3) 채널 변경

setChannel) self.channel = 11

getChannel) self.channel = 11

4) 볼륨 변경

setVolume) self.volume = 20

getVolume) self.volume = 20

5) 전원 끄기

setOn) self.on = False

getOn) self.on = False

>>>

```

    def setOn(self, on):
        self.on = on
        print("setOn) self.on =", self.on)

#main
print("1) main start")
print("2) 객체 t 생성")
t = Television(9, 10, True)
t.show()

print("3) 채널 변경")
t.setChannel(11)
t.getChannel()

print("4) 볼륨 변경")
t.setVolume(20)
t.getVolume()

print("5) 전원끄기")
t.setOn(False)
t.getOn()

```

```

1) main start
2) 객체 t 생성
In init..
init) self.channel = 9
      self.volume = 10
      self.on = True
show) self.channel = 9
      self.volume = 10
      self.on = True
3) 채널 변경
setChannel) self.channel = 11
getChannel) self.channel = 11
4) 볼륨 변경
setVolume) self.volume = 20
getVolume) self.volume = 20
5) 전원끄기
setOn) self.on = False
getOn) self.on = False
>>>

```

self

- 하나의 class에서 여러 객체를 생성 가능
- 현재의 객체를 가리키는 기능을 하는 지시어. 즉 method가 어떤 instance에서 호출됐는지 알려주는 인스턴스 참조자 (instance reference)

```
class Ball:
    def bounce(self):
        if self.direction == "down":
            self.direction = "up"
```

```
myBall = Ball()
yourBall = Ball()
```

- 위의 예에서 bounce()라는 method입장에서는 어느 instance가 자신을 호출했는 지 알아야 함
- self 인자는 어느 객체가 method를 호출했는 지 알려줌
→ 이를 instance reference라 함.

Object⁰¹이 method를 호출할 때 어떤 instance가 호출했는지 instance reference를 method로 자동으로 넘겨줌

```

1 class Ball:
2
3     size = 0
4     direction = "default"
5
6     def __init__(self, in_size, in_direction) :
7         print("i-1) In init, self =", self)
8         self.size = in_size
9         self.direction = in_direction
10
11         print("i-2) self.size =", self.size)
12         print("i-3) self.dircion =", self.direction, "\n")
13
14     def bounce(self):
15         print("b-1) self = ", self)
16         if self.direction == "down":
17             self.direction = "up"
18
19 # main
20 print("1) main \n")
21
22 print("2) myBall 생성")
23 myBall = Ball(10, "down")
24
25 print("3) yourBall 생성")
26 yourBall = Ball(20, "up")
27
28 print("4) call myBall.bounce()")
29 myBall.bounce()
30
31 print("\n5) call yourBall.bounce()")
32 yourBall.bounce()
33

```

1) main

2) myBall 생성

i-1) In init, self = <__main__.Ball object at 0x0368C5B0>

i-2) self.size = 10

i-3) self.dircion = down

3) yourBall 생성

i-1) In init, self = <__main__.Ball object at 0x036EC2C8>

i-2) self.size = 20

i-3) self.dircion = up

4) call myBall.bounce()

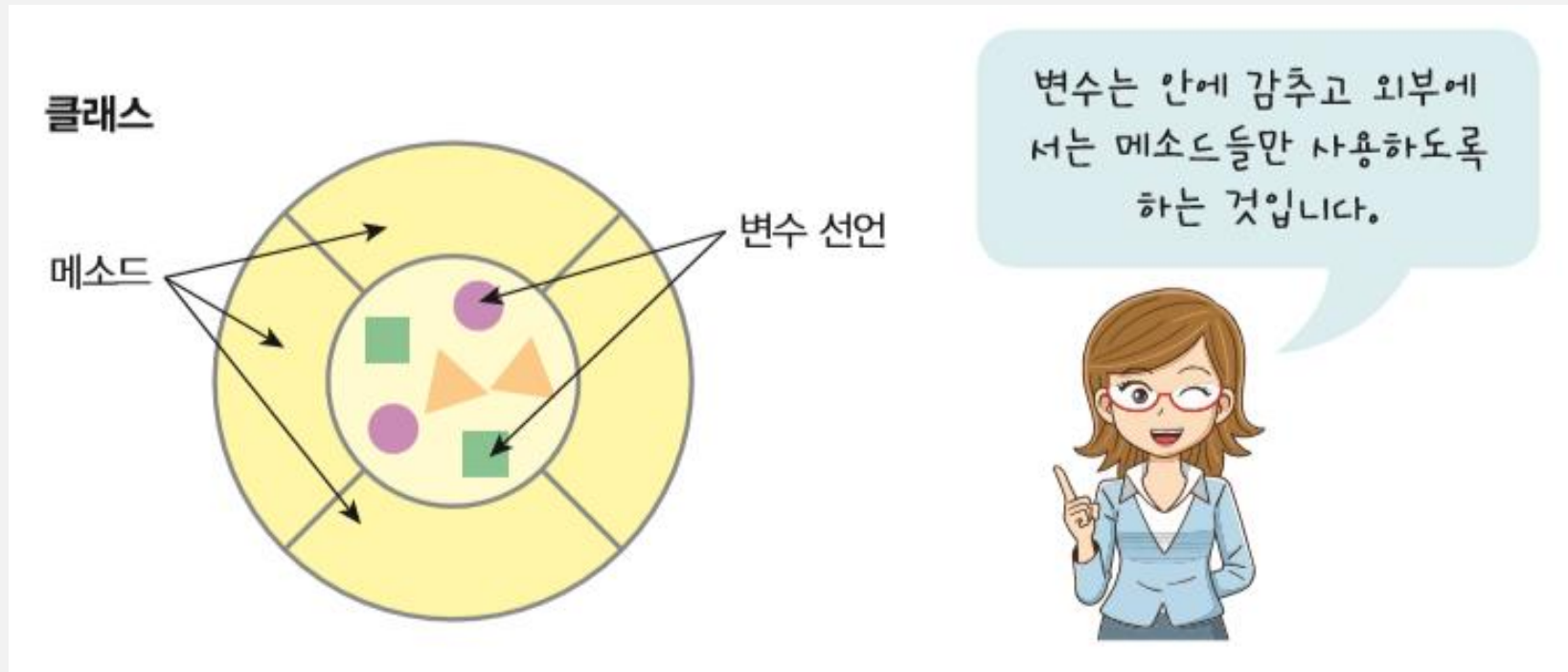
b-1) self = <__main__.Ball object at 0x0368C5B0>

5) call yourBall.bounce()

b-1) self = <__main__.Ball object at 0x036EC2C8>

정보은닉(Information Hiding)

- 구현의 세부 사항을 Class 안에 감추는 것



Private instance variable

- Instance variable을 **private**으로 선언하여 사용하는 것을 권장
 - 선언하고자 하는 변수 앞에 **__** (underline 2개)를 붙이면 **private**으로 선언됨
- Method도 이름 앞에 **__**를 붙이면 **private method**가 됨

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

obj = Student()
print(obj.__age)
```

```
...
AttributeError: 'Student' object has no attribute '__age'
```

```

1 class Student:
2     def __init__(self, name=None, age=0):
3         print("\ni-1) In init, name과 age를 private 선언")
4         self.__name = name
5         self.__age = age
6
7         print("i-2) self.__name =", self.__name)
8         print("        self.__age =", self.__age)
9
10    #main
11    print("1) main start")
12    print("2) 객체 obj 생성")
13    obj = Student("Park", 20)
14
15    print("\n3) private 선언된 __age 값을 직접 접근시 error 발생")
16    print(obj.__age)
17
18

```

1) main start
2) 객체 obj 생성

i-1) In init, name과 age를 private 선언
i-2) self.__name = Park
self.__age = 20

3) private 선언된 __age 값을 직접 접근시 error 발생

Traceback (most recent call last):

File "C:\₩과소사\₩과소사-강의예제\student.py", line 16, in <module>
 print(obj.__age)

AttributeError: 'Student' object has no attribute '__age'

^^^

접근자와 설정자

- **Private** 선언된 변수를 읽어오고 변수값을 변경하는 방법
 - 접근자(getter, accessor) : instance 변수값 (attributes)을 반환 (**read only**)
 - 설정자(setter, mutator) : instance 변수값(attributes)을 설정하는 **변경(write)**



접근자와 설정자

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

    def getAge(self):
        return self.__age

    def getName(self):
        return self.__name

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

    def setName(self, name):
        self.__name=name

obj=Student("Hong", 20)
obj.getName()
```

```

1 class Student:
2     def __init__(self, name=None, age=0):
3         print("i-1) name과 age를 private 선언")
4         self.__name = name
5         self.__age = age
6         print("i-2) self.__name =", self.__name)
7         print("      self.__age =", self.__age)
8
9     def getAge(self):
10        print("getAge) self.__age =", self.__age)
11        return self.__age
12
13    def getName(self):
14        print("getName) self.__name =", self.__name)
15        return self.__name
16
17    def setAge(self, age):
18        self.__age=age
19        print("setAge) self.__age =", self.__age)
20
21    def setName(self, name):
22        self.__name= name
23        print("setName) self.__name =", self.__name)
24
25 #main
26 print("1) main start")
27 print("2) 객체 obj 생성")
28 obj = Student("Park", 20)
29
30 # error !!
31 #print(obj.__age)
32
33 print("3) obj.__name =", obj.getName())
34 print("4) obj.__age =",obj.getAge())
35
36 print("5) obj 객체값 변경")
37 obj.setName("Hyun")
38 obj.setAge(25)
39
40 print("6) obj.__name =", obj.getName())
41 print("7) obj.__age =",obj.getAge())
42

```

```

1) main start
2) 객체 obj 생성
i-1) name과 age를 private 선언
i-2) self.__name = Park
      self.__age = 20
getName) self.__name = Park
3) obj.__name = Park
getAge) self.__age = 20
4) obj.__age = 20
5) obj 객체값 변경
setName) self.__name = Hyun
setAge) self.__age = 25
getName) self.__name = Hyun
6) obj.__name = Hyun
getAge) self.__age = 25
7) obj.__age = 25

```

Lab: 원을 Class로 표현

- 원을 Class도 표시해보자.
- 원은 반지름(radius)을 가지고 있다.
- 원의 넓이와 둘레를 계산하는 Method도 정의해보자.
설정자와 접근자 Method도 작성한다.

```

1 import math
2
3 class Circle:
4
5     def __init__(self, radius):
6
7         if radius <= 0:
8             print("Cl-1) 반지름 값이 0 보다 작습니다.")
9             exit()
10        else:
11            self.__Pie = math.pi
12            self.__radius = radius
13            print("Cl-2) self.__radius =", self.__radius)
14            print("        self.__Pie =", self.__Pie)
15
16    def setRadius(self, r):
17        self.__radius = r
18
19    def getRadius(self):
20        return self.__radius
21
22    def calcArea(self):
23        area = self.__Pie * self.__radius * self.__radius
24        return area
25
26    def calcCircum(self):
27        circumference = 2.0 * self.__Pie * self.__radius
28        return circumference
29
30 class Radius_input:
31
32     def __init__(self, r = 1.0):
33         self.radius = r
34         print("Ra_init) r =", r)
35
36     def in_value(self):
37         self.radius = float(input(">>> 원의 반지름을 입력하시오 :"))
38         print(">> self.radius =", self.radius)
39         return self.radius
40

```

```

41 #main
42 rad = Radius_input()
43
44 r = rad.in_value()
45 print("1) r =", r)
46
47 c1 = Circle(r)
48
49 print("\n2) 원의 반지름=", c1.getRadius())
50 print("3) 원의 넓이=", c1.calcArea())
51 print("4) 원의 둘레=", c1.calcCircum())
52

```

```

Ra_init) r = 1.0
>>> 원의 반지름을 입력하시오 :5
>> self.radius = 5.0
1) r = 5.0
Cl-2) self.__radius = 5.0
        self.__Pie = 3.141592653589793

2) 원의 반지름= 5.0
3) 원의 넓이= 78.53981633974483
4) 원의 둘레= 31.41592653589793

```

Lab: 은행 계좌

- 우리는 은행 계좌에 돈을 저금할 수 있고 인출할 수도 있다. 은행 계좌를 Class로 모델링하여 보자. 은행 계좌는 현재 잔액(balance)만을 인스턴스 변수로 가진다. 생성자와 인출 Method `withdraw()`와 저축 Method `deposit()` 만을 가정하자.

통장에서 **100** 가 출금되었음
통장에 **10** 가 입금되었음

Solution

```
class BankAccount:
    def __init__(self):
        self.__balance = 0

    def withdraw(self, amount):
        self.__balance -= amount
        print("통장에서 ", amount, "가 출금되었음 ")
        return self.__balance

    def deposit(self, amount):
        self.__balance += amount
        print("통장에 ", amount, "가 입금되었음")
        return self.__balance

a = BankAccount()
a.deposit(100)
a.withdraw(10)
```

Lab: 고양이 Class

- 고양이를 Class로 정의. 고양이는 이름(name)과 나이(age)를 속성으로 가진다.



```
Missy 3  
Lucky 5
```

Solution

```
class Cat:
    def __init__(self, name, age):
        self.__name = name
        self.__age = age

    def setName(self, name):
        self.__name = name

    def getName(self):
        return self.__name

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

    def getAge(self):
        return self.__age

missy = Cat('Missy', 3)
lucky = Cat('Lucky', 5)

print (missy.getName(), missy.getAge())
print (lucky.getName(), lucky.getAge())
```

Lab: 객체 생성과 사용

- 상자를 나타내는 Box Class를 작성하여 보자. Box Class는 가로길이, 세로길이, 높이를 나타내는 인스턴스 변수를 가진다.

(100, 100, 100)

상자의 부피는 1000000

Solution

```
class Box:
    def __init__(self, width=0, length=0, height=0):
        self.__width = width
        self.__length = length
        self.__height = height

    def setWidth(self, width):
        self.__width = width;

    def setLength(self, length):
        self.__length = length;

    def setHeight(self, height):
        self.__height = height;

    def getVolume(self):
        return self.__width*self.__length*self.__height

    def __str__(self):
        return '(%d, %d, %d)' % (self.__width, self.__length,
self.__height)

box = Box(100, 100, 100)
print(box)
print('상자의 부피는 ', box.getVolume())
```

Lab: 자동차 Class 작성

- 자동차를 나타내는 Class를 정의하여 보자. 예를 들어, 자동차 객체의 경우, 속성은 색상, 현재 속도, 현재 기어 등이다. 자동차의 동작은 기아 변속하기, 가속하기, 감속하기 등을 들 수 있다. 이 중에서 다음 그림과 같은 속성과 동작만을 추려서 구현해 보자.

```
(100, 3, white)
```

Solution

```
class Car:
    def __init__(self, speed=0, gear=1, color="white"):
        self.__speed = speed
        self.__gear = gear
        self.__color = color

    def setSpeed(self, speed):
        self.__speed = speed;

    def setGear(self, gear):
        self.__gear = gear;

    def setColor(self, color):
        self.__color = color;

    def __str__(self):
        return '(%d, %d, %s)' % (self.__speed, self.__gear, self.__color)

myCar = Car()
myCar.setGear(3);
myCar.setSpeed(100);
print(myCar)
```

객체를 함수로 전달할 때

- 객체를 argument로 지정하여 함수로 전달할 수 있음
 - User-defined 객체가 전달되면 함수가 이 객체를 변경할 수 있음
 - 문자열 등과 같은 system-defined 객체의 경우 이를 전달받은 함수는 이 객체를 변경할 수 없음

객체를 함수로 전달할 때

```
rectangle.py
File Edit Format Run Options Window Help
1 # 사각형을 클래스로 정의
2 count = 5
3
4 class Rectangle:
5     def __init__(self, side = 0):
6         self.side = side
7         print("init) self.side =", self.side)
8
9     def getArea(self):
10        return self.side*self.side
11
12 # 사각형 객체(r)와 반복횟수(n)를 받아서 변을 증가시키면서 면적을 출력
13 def printAreas(r, n):
14     print("nprintArea) 반복횟수 =", n)
15     while n >= 1:
16         print(">> r.side =", r.side, "Wt", "area = ", r.getArea())
17         r.side = r.side + 1
18         n = n - 1
19
20 # printAreas()을 호출하여서 객체의 내용이 변경되는지를 확인한다.
21 print("1) main")
22 myRect = Rectangle();
23
24 printAreas(myRect, count)
25 print("Wn2) end")
26
```

```
1) main
init) self.side = 0

printArea) 반복횟수 = 5
>> r.side = 0      area = 0
>> r.side = 1      area = 1
>> r.side = 2      area = 4
>> r.side = 3      area = 9
>> r.side = 4      area = 16

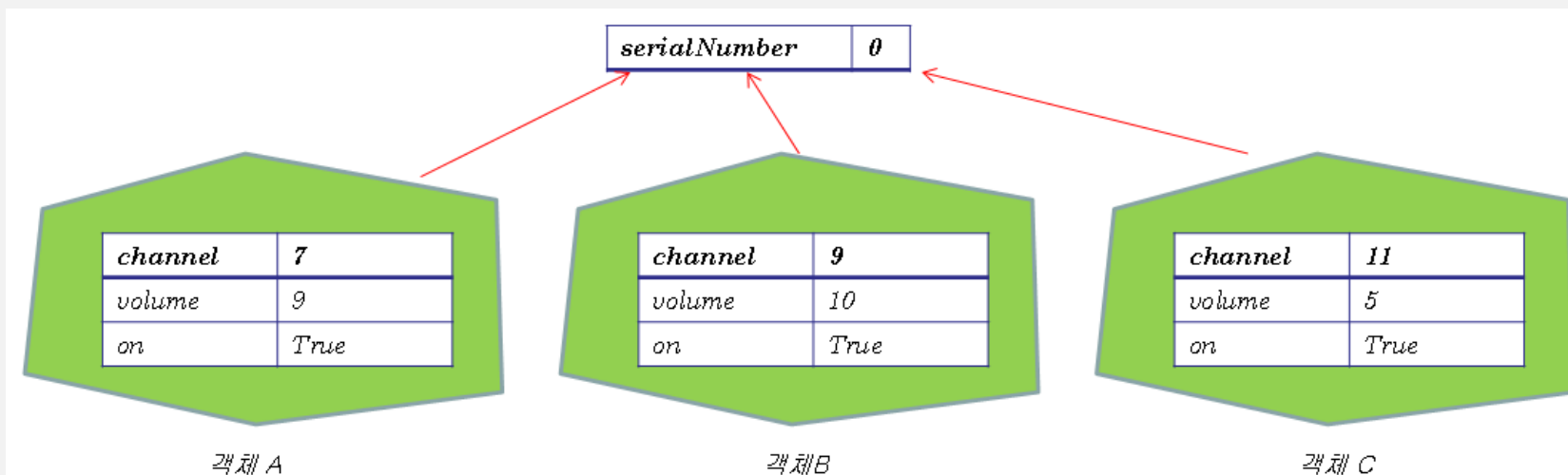
2) end
```

Class 변수(static 변수)

- 하나의 class에서 생성된 (여러) object(instance)들이 상호공유하는 변수
- 이들 변수는 모든 object(instance)를 통틀어서 unique해야 함. 모든 객체가 이것을 공유
- 이러한 변수를 정적변수(static member, class member, static variable)라고 함

Class 변수(static 변수)

```
class Television:
    serialNumber = 0          # 이것이 정적 변수이다.
    def __init__(self):
        Television.serialNumber += 1
    self.number = Television.serialNumber
    ...
```



```
1 class Television:
2
3     count = 0          # class variable
4     serial_number = 100 # class variable
5
6     def __init__(self, channel, volume, on):
7         self.channel = channel
8         self.volume = volume
9         self.on = on
10
11         Television.count += 1
12         print("init) Television.count =", Television.count)
13
14         Television.serial_number += 10
15         print("init) Television.serial_number =", Television.serial_number)
16
17     def __getCount__(self):
18 #         print("g-1) self.count =", self.count)
19         return self.count
20
21     def __getSerial_number__(self):
22         print("g-1) self.serial_number =", self.serial_number)
23         return self.serial_number
24
25
```

```
1) 객체 t1 생성
init) Television.count = 1
init) Television.serial_number = 110

2) Television.count = 1
   Television.serial_number = 110

3) 객체 t2 생성
init) Television.count = 2
init) Television.serial_number = 120

4) Television.count = 2
   Television.serial_number = 120

5) 객체 t3 생성
init) Television.count = 3
init) Television.serial_number = 130

6) Television.count = 3
   Television.serial_number = 130

7) Class 변수 serial_number 값을 200으로 변경
8) Television.serial_number = 200

9) 객체 t4 생성
init) Television.count = 4
init) Television.serial_number = 210

10) Television.count = 4
    Television.serial_number = 210

11) t1.count = 4
12) t2.count = 4
13) t3.count = 4
14) t4.count = 4

15) t1.count = 4
    t2.count = 4
    t3.count = 4
    t4.count = 4
```

```

26 #main
27 print("1) 객체 t1 생성")
28 t1 = Television(9, 10, True)
29 print("Wn2) Television.count =", Television.count)
30 print("    Television.serial_number =", Television.serial_number)
31
32 print("Wn3) 객체 t2 생성")
33 t2 = Television(11, 12, True)
34 print("Wn4) Television.count =", Television.count)
35 print("    Television.serial_number =", Television.serial_number)
36
37 print("Wn5) 객체 t3 생성")
38 t3 = Television(13, 15, False)
39 print("Wn6) Television.count =", Television.count)
40 print("    Television.serial_number =", Television.serial_number)
41
42 print("Wn7) Class 변수 serial_number 값을 200으로 변경")
43 Television.serial_number = 200
44 print("8) Television.serial_number = ", Television.serial_number)
45
46 print("Wn9) 객체 t4 생성")
47 t4 = Television(12, 20, False)
48 print("Wn10) Television.count =", Television.count)
49 print("    Television.serial_number =", Television.serial_number)
50
51 print("Wn11) t1.count =", t1.__getCount__())
52 print("12) t2.count =", t2.__getCount__())
53 print("13) t3.count =", t3.__getCount__())
54 print("14) t4.count =", t4.__getCount__())
55
56 print("Wn15) t1.count =", t1.count)
57 print("    t2.count =", t2.count)
58 print("    t3.count =", t3.count)
59 print("    t4.count =", t4.count)

```

```

1) 객체 t1 생성
init) Television.count = 1
init) Television.serial_number = 110

2) Television.count = 1
   Television.serial_number = 110

3) 객체 t2 생성
init) Television.count = 2
init) Television.serial_number = 120

4) Television.count = 2
   Television.serial_number = 120

5) 객체 t3 생성
init) Television.count = 3
init) Television.serial_number = 130

6) Television.count = 3
   Television.serial_number = 130

7) Class 변수 serial_number 값을 200으로 변경
8) Television.serial_number = 200

9) 객체 t4 생성
init) Television.count = 4
init) Television.serial_number = 210

10) Television.count = 4
    Television.serial_number = 210

11) t1.count = 4
12) t2.count = 4
13) t3.count = 4
14) t4.count = 4

15) t1.count = 4
    t2.count = 4
    t3.count = 4
    t4.count = 4

```

Special method : 객체 출력 method

- Special method
 - class를 정의 시 Python이 자동으로 포함시키는 method들
- 그 중의 하나가 특수 method '__str__()'
 - 객체를 print로 출력할 때 Python이 어떤 내용으로 표시할 지 알려줌
 - 기본출력 내용
 - Instance가 정의된 곳 (앞의 예제에서는 __main__)
 - Class 이름 (앞의 예제에서는 Ball)
 - Instance가 저장되어있는 메모리상의 위치

Default __str__()

default_str_.py

File Edit Format Run Options Window Help

default __str__()를 이용하여 객체에 대한 기본정보를 출력

```
class Ball:
    def __init__(self, color, size, direction):
        print("Wninit) self =", self, "Wn")
        self.color = color
        self.size = size
        self.direction = direction
```

```
myBall = Ball("red", "small", "down")
print (myBall)
```

```
yourBall = Ball("blue", "large", "up")
print (yourBall)
```

init) self = <__main__.Ball object at 0x0316A370>

<__main__.Ball object at 0x0316A370>

init) self = <__main__.Ball object at 0x035CDC10>

<__main__.Ball object at 0x035CDC10>

Special method : 객체 출력 method

- 특수 method '`__str__()`'
 - Default로 보여주는 내용 이외에 개발자가 객체에 대해 정의한 내용으로도 출력가능

```
class Ball:
    def __init__(self, color, size, direction):
        self.color = color
        self.size = size
        self.direction = direction

    def __str__(self):
        msg = "Hi, I'm a " + self.size + " " + self.color + " ball!"
        return msg
```

```
myBall = Ball("red", "small", "down")
print(myBall)  ← __str__() method에서 반환되는 문자열 출력
```


Special method : 객체 출력 method

```
user_defined__Str__.py
File Edit Format Run Options Window Help
1 # Using __str__() to change how the object prints
2
3 class Ball:
4
5     def __init__(self, color, size, direction):
6         self.color = color
7         self.size = size
8         self.direction = direction
9
10    # here's the special method __str__()
11    def __str__(self):
12        msg = "Hi, I'm a " + self.size + " " + self.color + " ball!"
13        print("s-1) msg =", msg)
14
15        return msg
16
17 # main
18
19 myBall = Ball("red", "small", "down")
20 print (myBall)
21
```

```
s-1) msg = Hi, I'm a small red ball!
Hi, I'm a small red ball!
```

```

1 class HotDog:
2
3     def __init__(self):
4         self.cooked_level = 0
5         self.cooked_string = "Raw"
6         self.condiments = []
7
8         print("Whi-1) self.cooked_level =", self.cooked_level)
9         print("      self.cooked_string =", self.cooked_string)
10        print("      self.condiments =", self.condiments)
11
12    def __str__(self):
13        msg = "hot dog"
14
15        if len(self.condiments) > 0:
16            msg = msg + " with "
17
18        for i in self.condiments:
19            msg = msg + i + ", "
20            print("s-1) msg =", msg)
21
22        msg = msg.strip(", ") # 마지막 ", " 삭제
23        print("s-2) msg =", msg)
24
25        msg = self.cooked_string + " " + msg + "."
26        print("s-3) 생성된 메시지 :", msg)
27        return msg
28
29    def cook(self, time):
30        print("c-1) self.cooked_level = {}, time = {}".format(self.cooked_level, time))
31        self.cooked_level = self.cooked_level + time
32        print("c-2) in cook(), 요청된 굽는 시간 :", self.cooked_level)
33
34        if self.cooked_level > 8:
35            self.cooked_string = "Charcoal"
36        elif self.cooked_level > 5:
37            self.cooked_string = "Well-done"
38        elif self.cooked_level > 3:
39            self.cooked_string = "Medium"
40        else:
41            self.cooked_string = "Raw"
42
43        print("c-3) in cook(), cooked_string = ", self.cooked_string)
44
45    def addCondiment(self, stuff):
46        print("Wha-1) 추가된 양념 =", stuff)
47        self.condiments.append(stuff)
48        print("a-2) 최종 양념 =", self.condiments)
49

```

1) main start.myDoc객체 생성

```

i-1) self.cooked_level = 0
     self.cooked_string = Raw
     self.condiments = []

```

2) myDog 객체 정보 출력

```

s-2) msg = hot dog
s-3) 생성된 메시지 : Raw hot dog.
Raw hot dog.

```

3) Cooking hot dog for 4 minutes.

```

c-1) self.cooked_level = 0, time = 4
c-2) in cook(), 요청된 굽는 시간 : 4
c-3) in cook(), cooked_string = Medium

```

4) myDog 객체 정보 출력

```

s-2) msg = hot dog
s-3) 생성된 메시지 : Medium hot dog.
Medium hot dog.

```

5) Cooking hot dog for 3 more minutes.

```

c-1) self.cooked_level = 4, time = 3
c-2) in cook(), 요청된 굽는 시간 : 7
c-3) in cook(), cooked_string = Well-done

```

6) myDog 객체 정보 출력

```

s-2) msg = hot dog
s-3) 생성된 메시지 : Well-done hot dog.
Well-done hot dog.

```

7) What happens if I cook it for 10 more minutes?

```

c-1) self.cooked_level = 7, time = 10
c-2) in cook(), 요청된 굽는 시간 : 17
c-3) in cook(), cooked_string = Charcoal

```

8) myDog 객체 정보 출력

```

s-2) msg = hot dog
s-3) 생성된 메시지 : Charcoal hot dog.
Charcoal hot dog.

```

9) Now, I'm going to add some stuff on my hot dog

```

a-1) 추가된 양념 = ketchup
a-2) 최종 양념 = ['ketchup']

```

```

a-1) 추가된 양념 = mustard
a-2) 최종 양념 = ['ketchup', 'mustard']

```

10) myDog 객체 정보 출력
s-1) msg = hot dog with ketchup,
s-1) msg = hot dog with ketchup, mustard,
s-2) msg = hot dog with ketchup, mustard
s-3) 생성된 메시지 : Charcoal hot dog with ketchup, mustard.
Charcoal hot dog with ketchup, mustard.

```

50 # main
51 print("1) main start.myDoc객체생성")
52 myDog = HotDog()
53 print("\n2) myDog 객체 정보 출력")
54 print(myDog)
55
56 print("\n3) Cooking hot dog for 4 minutes.")
57 myDog.cook(4)
58
59 print("\n4) myDog 객체 정보 출력")
60 print (myDog)
61
62 print ("\n5) Cooking hot dog for 3 more minutes.")
63 myDog.cook(3)
64
65 print ("\n6) myDog 객체 정보 출력")
66 print (myDog)
67
68 print ("\n7) What happens if I cook it for 10 more minutes?")
69 myDog.cook(10)
70 print ("\n8) myDog 객체 정보 출력")
71 print (myDog)
72
73 print ("\n9) Now, I'm going to add some stuff on my hot dog")
74 myDog.addCondiment("ketchup")
75 myDog.addCondiment("mustard")
76
77 print ("\n10) myDog 객체 정보 출력")
78 print (myDog)
79

```

1) main start.myDoc객체생성

```

i-1) self.cooked_level = 0
    self.cooked_string = Raw
    self.condiments = []

```

2) myDog 객체 정보 출력

```

s-2) msg = hot dog
s-3) 생성된 메시지 : Raw hot dog.
Raw hot dog.

```

3) Cooking hot dog for 4 minutes.

```

c-1) self.cooked_level = 0, time = 4
c-2) in cook(), 요청된 굽는 시간 : 4
c-3) in cook(), cooked_string = Medium

```

4) myDog 객체 정보 출력

```

s-2) msg = hot dog
s-3) 생성된 메시지 : Medium hot dog.
Medium hot dog.

```

5) Cooking hot dog for 3 more minutes.

```

c-1) self.cooked_level = 4, time = 3
c-2) in cook(), 요청된 굽는 시간 : 7
c-3) in cook(), cooked_string = Well-done

```

6) myDog 객체 정보 출력

```

s-2) msg = hot dog
s-3) 생성된 메시지 : Well-done hot dog.
Well-done hot dog.

```

7) What happens if I cook it for 10 more minutes?

```

c-1) self.cooked_level = 7, time = 10
c-2) in cook(), 요청된 굽는 시간 : 17
c-3) in cook(), cooked_string = Charcoal

```

8) myDog 객체 정보 출력

```

s-2) msg = hot dog
s-3) 생성된 메시지 : Charcoal hot dog.
Charcoal hot dog.

```

9) Now, I'm going to add some stuff on my hot dog

```

a-1) 추가된 양념 = ketchup
a-2) 최종 양념 = ['ketchup']

```

```

a-1) 추가된 양념 = mustard
a-2) 최종 양념 = ['ketchup', 'mustard']

```

10) myDog 객체 정보 출력

```

s-1) msg = hot dog with ketchup,
s-1) msg = hot dog with ketchup, mustard,
s-2) msg = hot dog with ketchup, mustard
s-3) 생성된 메시지 : Charcoal hot dog with ketchup, mustard.
Charcoal hot dog with ketchup, mustard.

```

Special method : 연산자 관련 method

- Python에는 연산자(+, -, *, /)에 관련된 **특수 메소드(special method)**가 있다.

```
class Circle:
    ...
    def __eq__(self, other):
        return self.radius == other.radius

c1 = Circle(10)
c2 = Circle(10)
if c1 == c2:
    print("원의 반지름은 동일합니다. ")
```

특수 Method

| 연산자 | 메소드 | 설명 |
|--------------|------------------------------------|-------------------------------|
| x + y | <code>__add__(self, y)</code> | 덧셈 |
| x - y | <code>__sub__(self, y)</code> | 뺄셈 |
| x * y | <code>__mul__(self, y)</code> | 곱셈 |
| x / y | <code>__truediv__(self, y)</code> | 실수나눗셈 |
| x // y | <code>__floordiv__(self, y)</code> | 정수나눗셈 |
| x % y | <code>__mod__(self, y)</code> | 나머지 |
| divmod(x, y) | <code>__divmod__(self, y)</code> | 실수나눗셈과 나머지 |
| x ** y | <code>__pow__(self, y)</code> | 지수 |
| x << y | <code>__lshift__(self, y)</code> | 왼쪽 비트 이동 |
| x >> y | <code>__rshift__(self, y)</code> | 오른쪽 비트 이동 |
| x <= y | <code>__le__(self, y)</code> | less than or equal(작거나 같다) |
| x < y | <code>__lt__(self, y)</code> | less than(작다) |
| x >= y | <code>__ge__(self, y)</code> | greater than or equal(크거나 같다) |
| x > y | <code>__gt__(self, y)</code> | greater than(크다) |
| x == y | <code>__eq__(self, y)</code> | 같다 |
| x != y | <code>__neq__(self, y)</code> | 같지않다 |

Special method : 연산자 관련 method

- `__add__()`

`__add__()` 연산을 2개 vector 값의 합을 구하는 method로 재정의

- 2차원 공간에서 vector는 (a, b)와 같이 2개의 실수로 표현

- +

$$(a, b) + (c, d) = (a + c, b + d)$$

- -

$$(a, b) - (c, d) = (a - c, b - d)$$

예제

```
class Vector2D :
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Vector2D(self.x + other.x, self.y + other.y)

    def __sub__(self, other):
        return Vector2D(self.x - other.x, self.y - other.y)

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __str__(self):
        return '(%g, %g)' % (self.x, self.y)

u = Vector2D(0,1)
v = Vector2D(1,0)
w = Vector2D(1,1)
a = u + v
print( a)
```

예제

```
vector.py
File Edit Format Run Options Window Help
1 class Vector2D :
2     def __init__(self, x, y):
3         self.x = x
4         self.y = y
5         print("init) self.x = ", self.x, "self.y = ", self.y)
6
7     def __add__(self, other):
8         print("add) self.x = ", self.x, "self.y = ", self.y)
9         print("add) other.x = ", other.x, "other.y = ", other.y)
10        return Vector2D(self.x + other.x, self.y + other.y)
11
12    def __sub__(self, other):
13        print("sub) self.x = ", self.x, "self.y = ", self.y)
14        print("sub) other.x = ", other.x, "other.y = ", other.y)
15        return Vector2D(self.x - other.x, self.y - other.y)
16
17    def __eq__(self, other):
18        print("equ) self.x = ", self.x, "self.y = ", self.y)
19        print("equ) other.x = ", other.x, "other.y = ", other.y)
20        return self.x == other.x and self.y == other.y
21
22    def __str__(self):
23        return '(%g, %g)' % (self.x, self.y)
24
25 #main
26 print("main) 객체 u 생성")
27 u = Vector2D(0,1)
28 print("1) print u = ", u, "\n")
29
30 print("2) 객체 v 생성")
31 v = Vector2D(1,0)
32 print("3) print v = ", v, "\n")
33
34 print("4) 객체 w 생성")
35 w = Vector2D(1,1)
36 print("5) print w = ", w, "\n")
37
38 print("6) 객체 add 생성, __add__ 호출")
39 add = u + v
40 print("7) print add = ", add, "\n")
41
42 print("8) 객체 sub 생성, __sub__ 호출")
43 sub = u - v
44 print("9) print sub = ", sub, "\n")
45
46 print("10) __eq__ 호출")
47 if u == v:
48     print("11) u == v")
49 else:
50     print("12) u != v")
51
```

main) 객체 u 생성
init) self.x = 0 self.y = 1
1) print u = (0, 1)

2) 객체 v 생성
init) self.x = 1 self.y = 0
3) print v = (1, 0)

4) 객체 w 생성
init) self.x = 1 self.y = 1
5) print w = (1, 1)

6) 객체 add 생성, __add__ 호출
add) self.x = 0 self.y = 1
add) other.x = 1 other.y = 0
init) self.x = 1 self.y = 1
7) print add = (1, 1)

8) 객체 sub 생성, __sub__ 호출
sub) self.x = 0 self.y = 1
sub) other.x = 1 other.y = 0
init) self.x = -1 self.y = 1
9) print sub = (-1, 1)

10) __eq__ 호출
equ) self.x = 0 self.y = 1
equ) other.x = 1 other.y = 0
12) u != v

Python에서의 변수의 종류

- 지역 변수 – 함수 안에서 선언되는 변수
- 전역 변수 – 함수 외부에서 선언되는 변수
- 인스턴스 변수 – Class 안에 선언된 변수, 앞에 self.가 붙는다.

핵심 정리

- Class는 속성과 동작으로 이루어진다. 속성은 인스턴스 변수로 표현되고 동작은 Method로 표현된다.
- 객체를 생성하려면 생성자 Method를 호출한다. 생성자 Method는 `__init__()` 이름의 Method이다.
- 인스턴스 변수를 정의하려면 생성자 Method 안에서 `self.변수이름` 과 같이 생성한다.

Q & A

