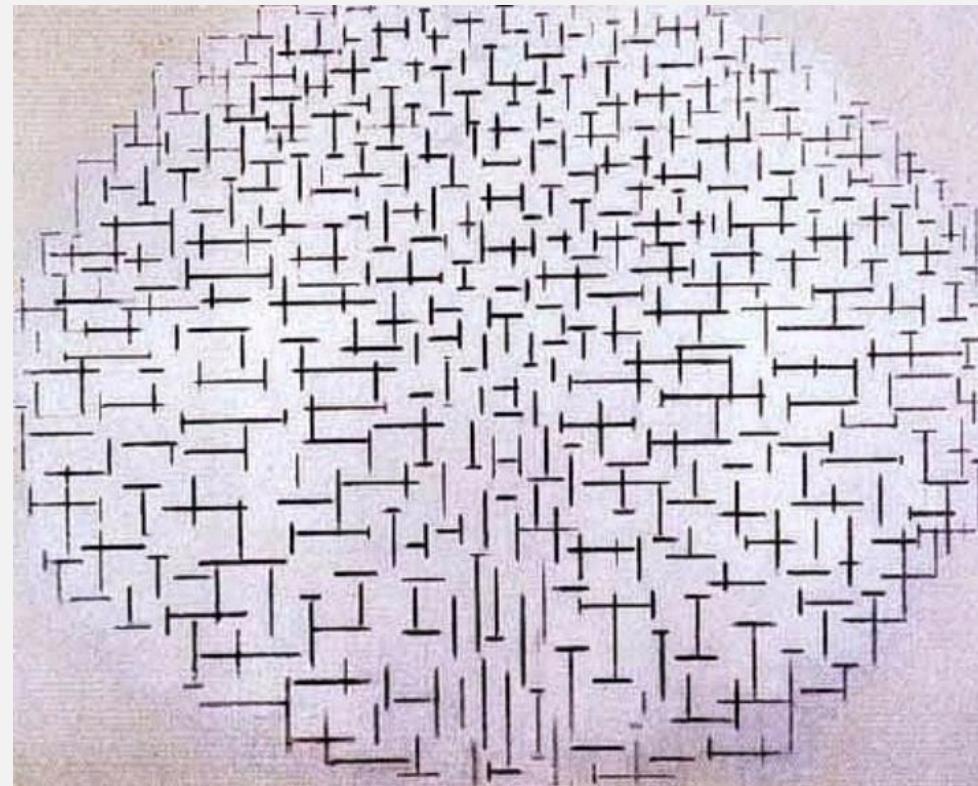


Class와 Object

By 박수현

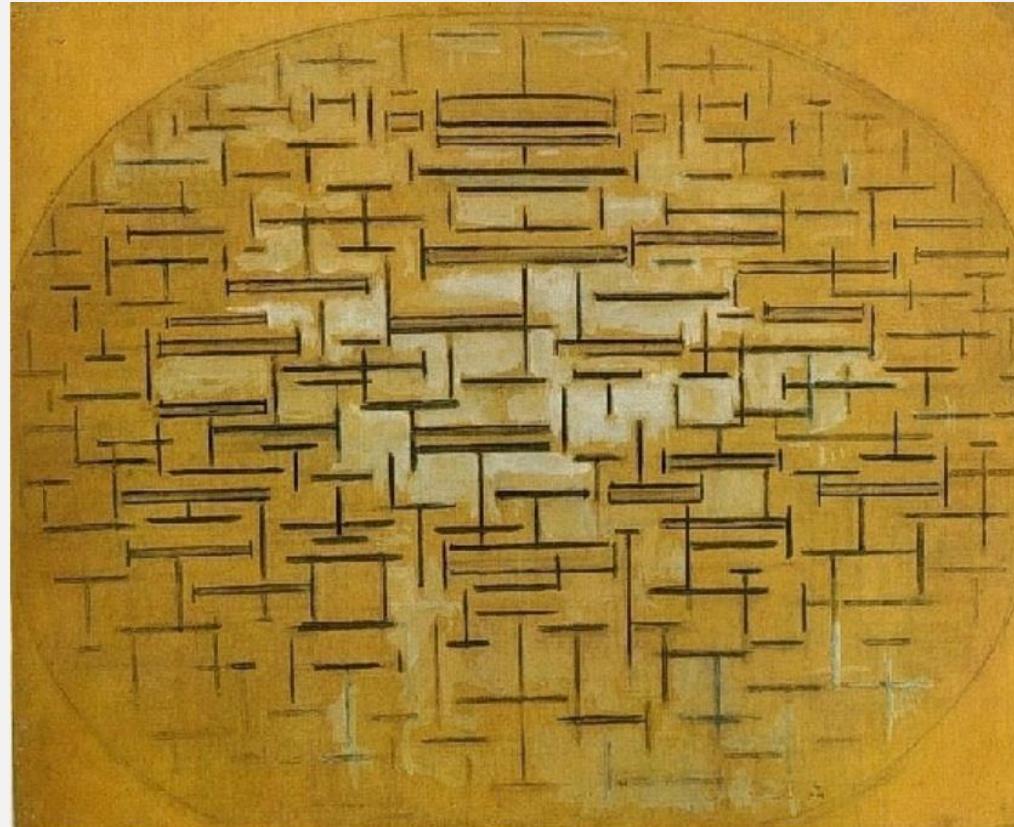
Programming Paradigm

- 다음 그림은 무엇을 그린 것일까 ?



Programming Paradigm

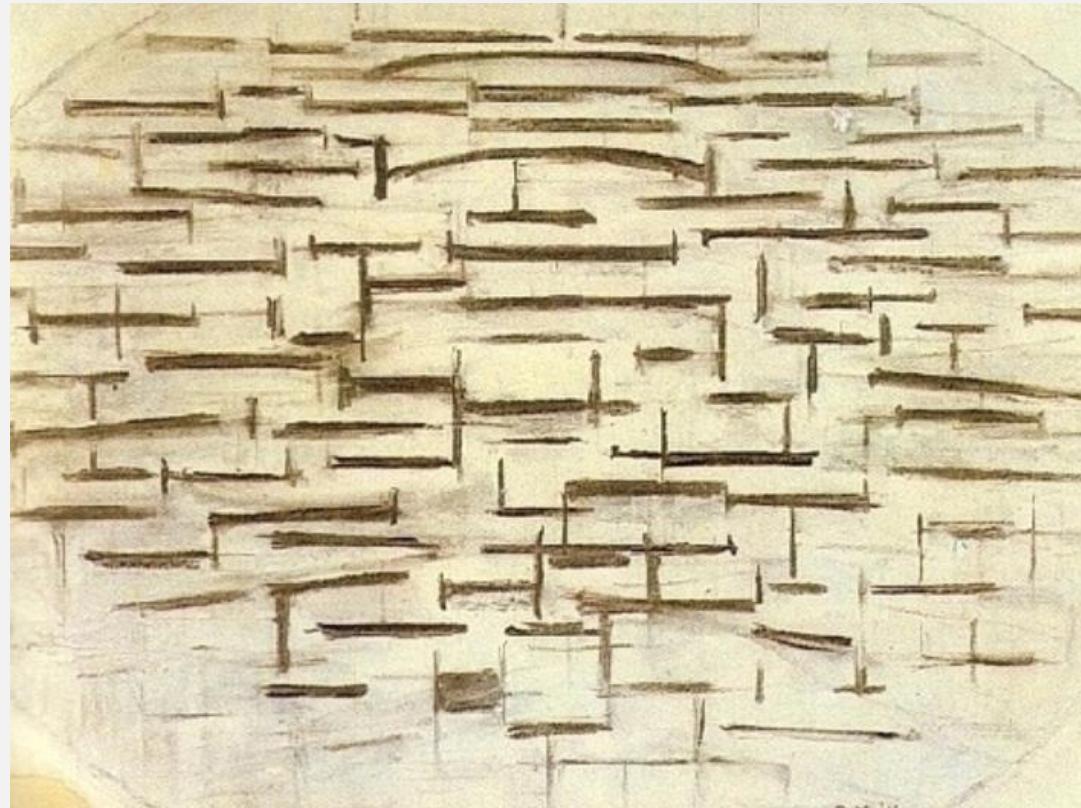
- 다음 그림은 무엇을 그린 것일까 ?



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

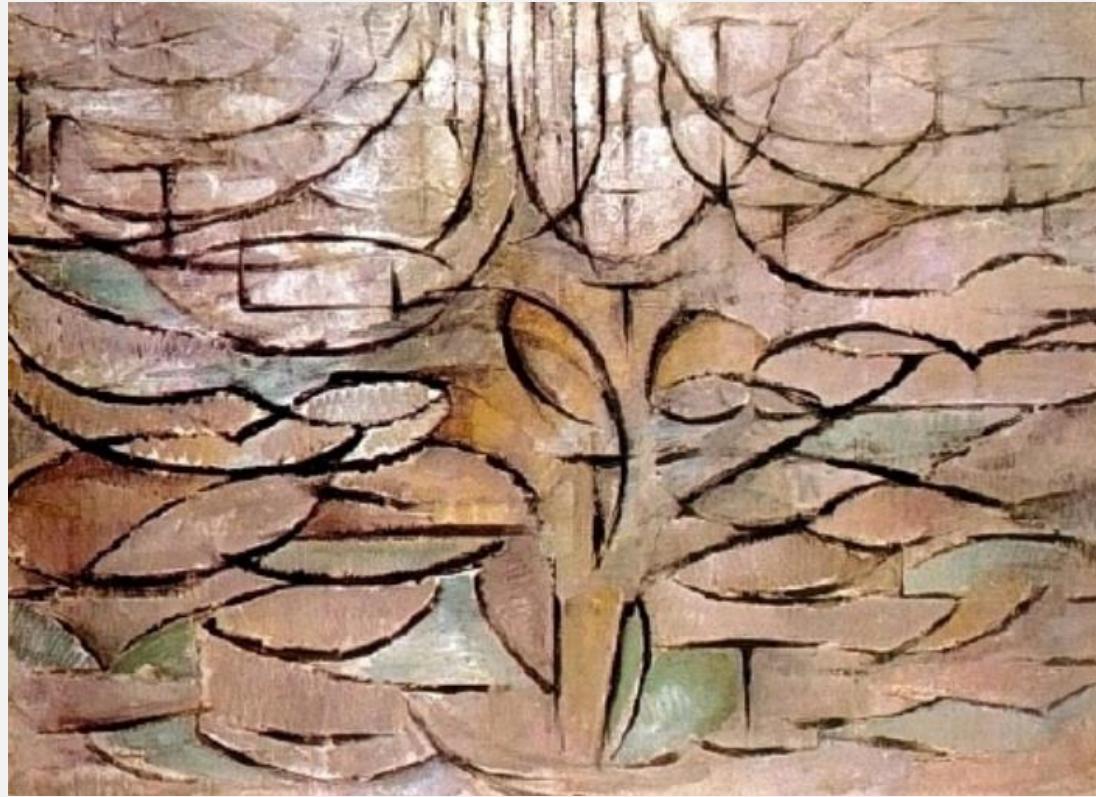
Programming Paradigm

- 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 ?



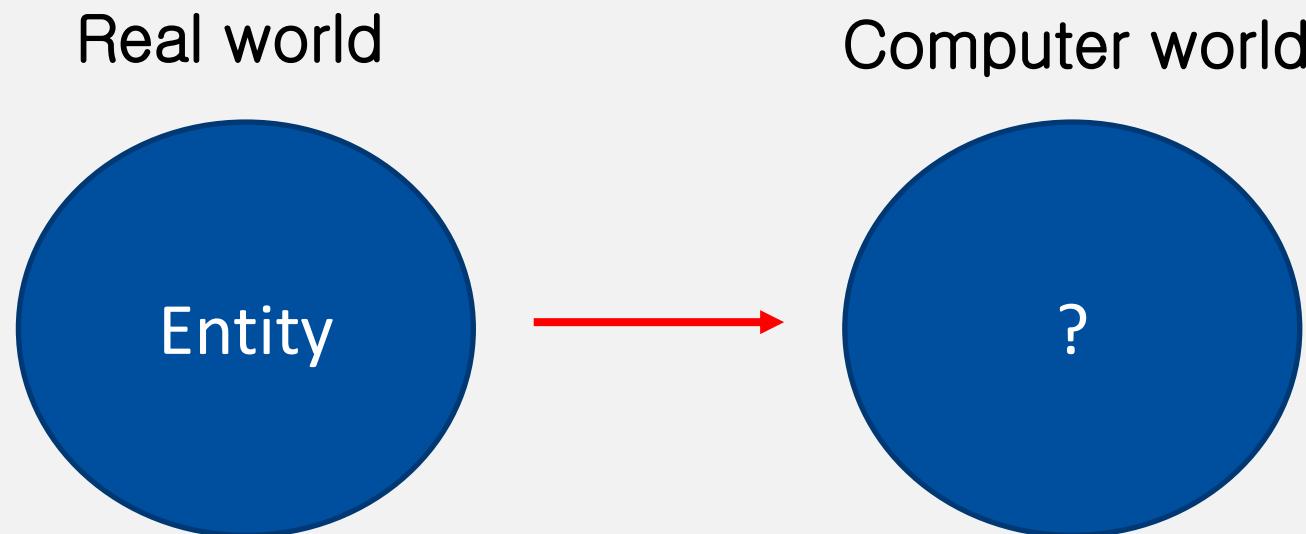
<http://blog.naver.com/PostView.nhn?blogId=jhinju&logNo=10185034539&parentCategoryNo=&categoryNo=38&viewDate=&isShowPopularPosts=true&from=search>

Programming Paradigm

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

Programming Paradigm

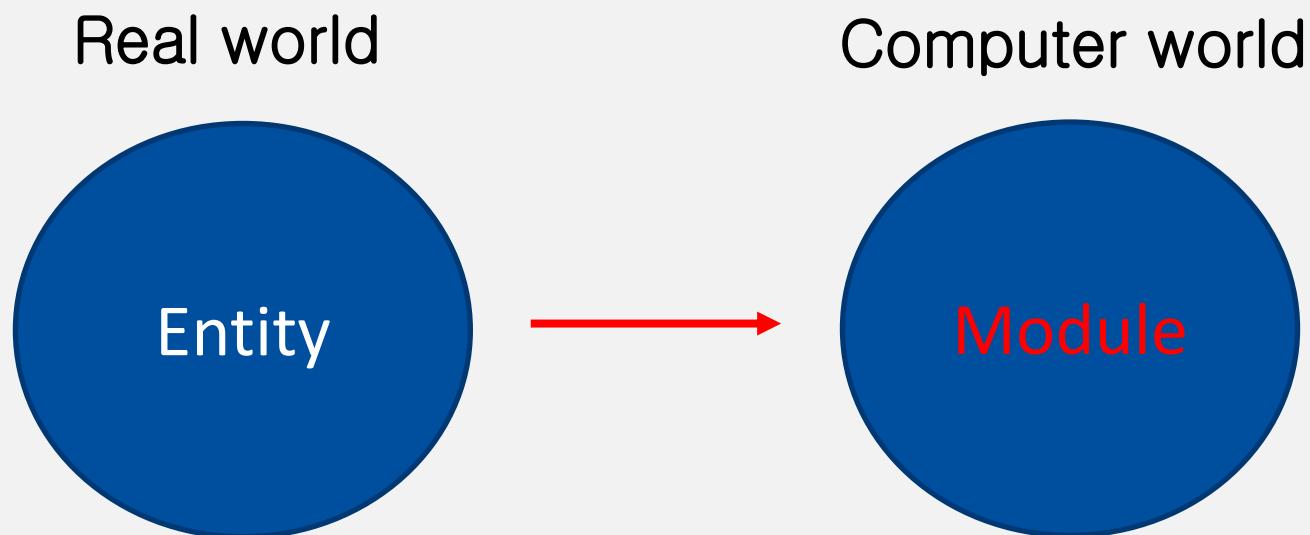
- Mapping / Modeling / 추상화 (抽象化, abstraction)



Programming Paradigm

- **Structure Programming (SP)**

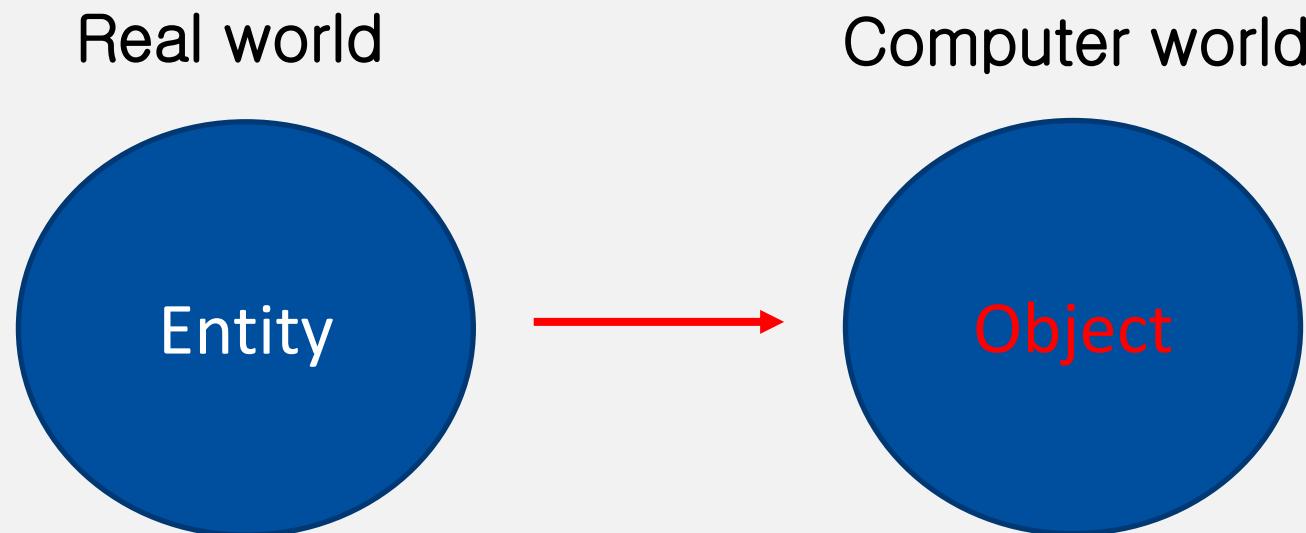
- Module
- Structure 기반
- Readability (가독성 : 可讀性)



Programming Paradigm

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

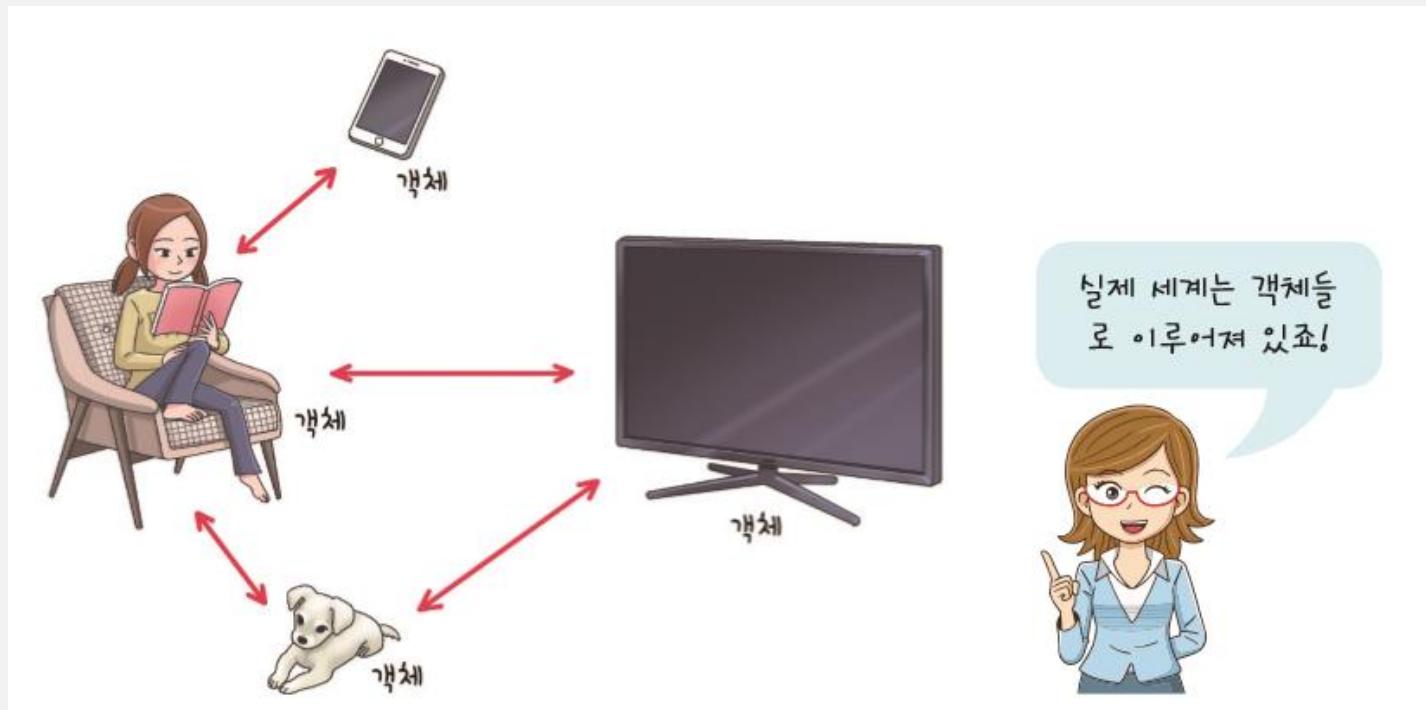
- Object
- Encapsulation (from information hiding)
- Reusability (재사용성 : 再使用性)



객체지향 프로그래밍

- Object-Oriented Programming (OOP)

Software를 객체(object)로 구성하는 programming paradigm

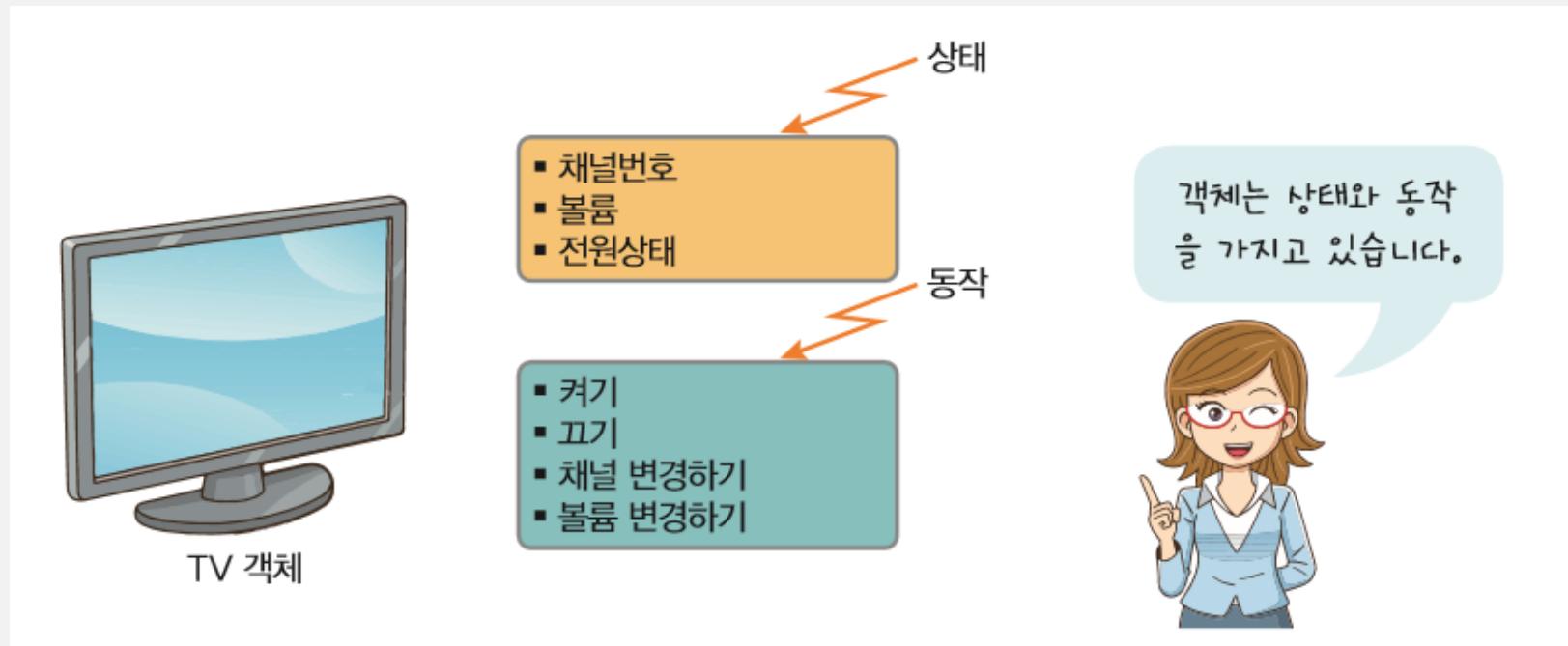


객체 (Object)

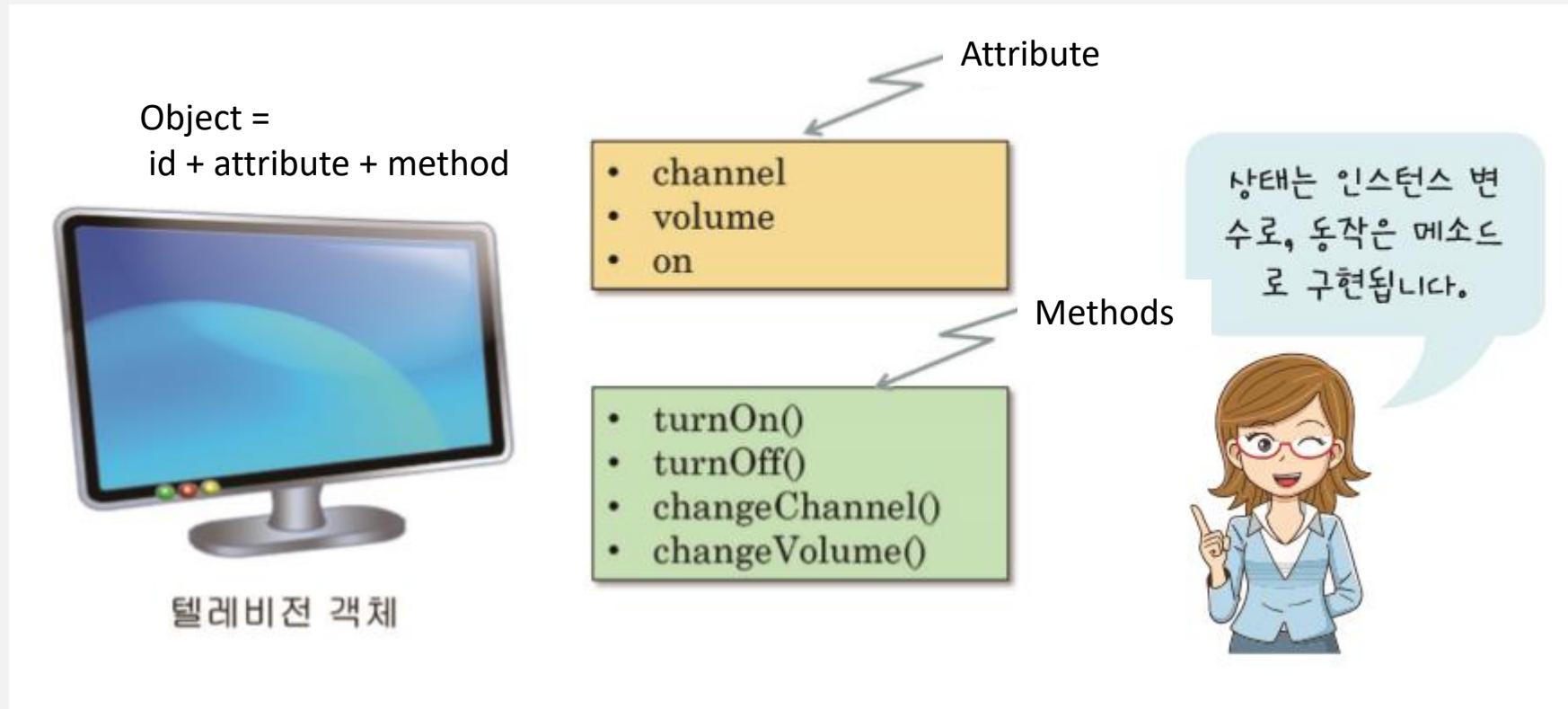
- 객체(Object) : 개체(entity)를 구성(정의)하는 속성(attribute, data)과 그 속성을 읽거나 변경(read / write)할 수 있는 operation (function)을 모아놓은 것.
 - Object = id + attribute + method
 - Object의 다른 이름 : 인스턴스(instance)
- Attribute (member variable)
 - 개체(entity)에서 **다루고자 하는 Data (or field)**
- Method (member function)
 - 속성을 읽거나 변경(read / write)할 수 있는 operation (function)
 - **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)



```

1 class Television: # class id
2
3     # attributes
4     channel = 0
5     volume = 0
6     on = "on"
7
8     #methods
9     def changeChannel(self, ch_num):
10        self.channel = ch_num
11
12     def changeVolume(self, volume_value):
13        self.volume = volume_value
14
15     def turnOn(self, on_off):
16        self.on = on_off
17
18     def display(self):
19         print("Wn>> channel :", self.channel)
20         print("    volume :", self.volume)
21         print("    on :" , self.on)
22
23 class Main:
24     def main(self):
25         tv = Television()
26
27         input_channel = int(input("Wn>> Channel number : "))
28         input_Volume = int(input("Wn>> Volume : "))
29         input_onoff = input("Wn>> on or off : ")
30
31         tv.changeChannel(input_channel)
32         tv.changeVolume(input_Volume)
33         tv.turnOn(input_onoff)
34
35         tv.display()
36
37 #main
38 m = Main()
39 m.main()

```

Object =
id + attribute + method



텔레비전 객체

- channel
- volume
- on

- turnOn()
- turnOff()
- changeChannel()
- changeVolume()

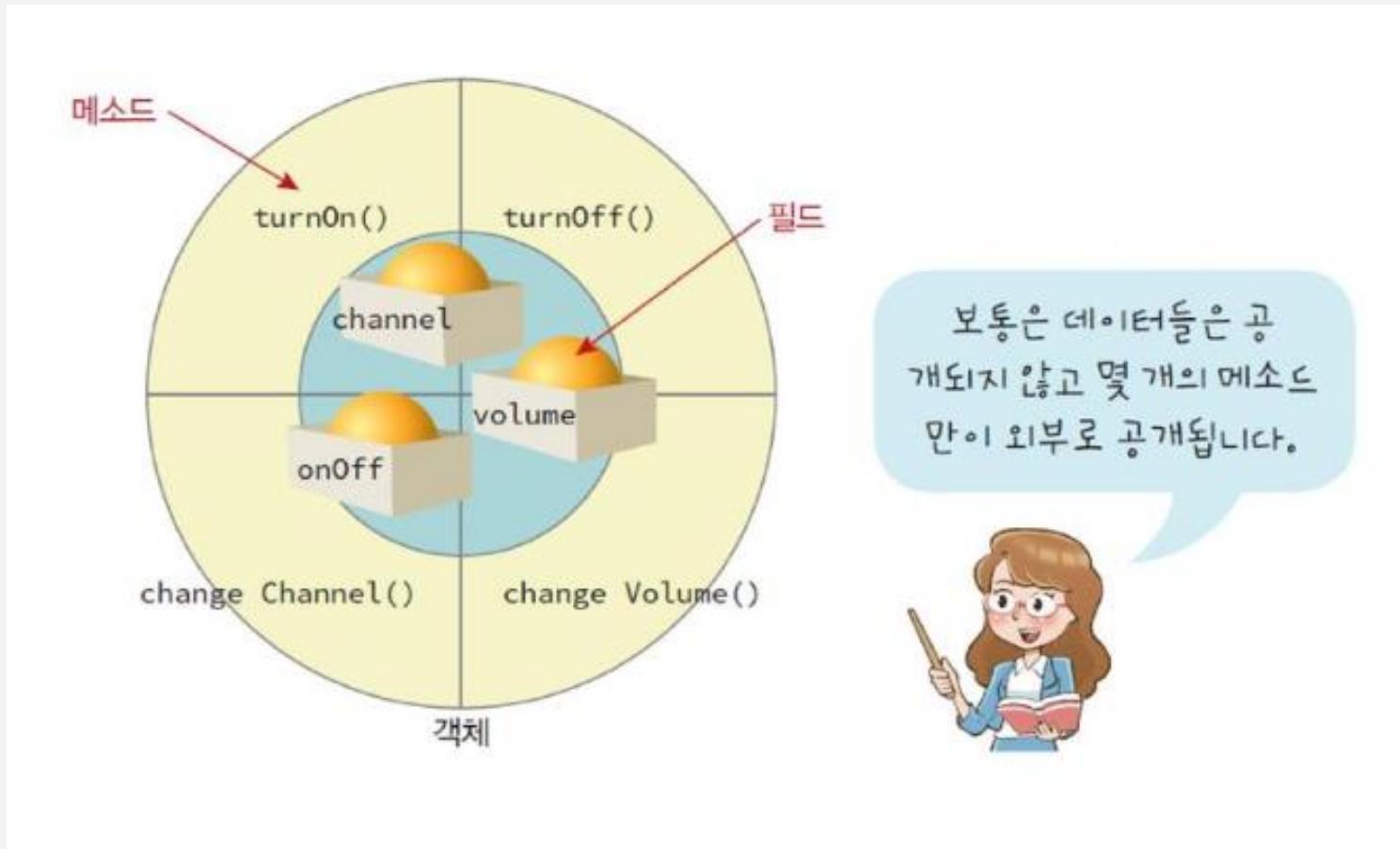
Attribute

Methods

상태는 인스턴스 변수로, 동작은 메소드로 구현됩니다.



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(modeling, abstraction)
- Basic philosophy of OOP : maximize the level of software **reusability**

- **Instantiation**

- Class(template)  objects (instances)
instantiation

- **Inheritance**

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



Class

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

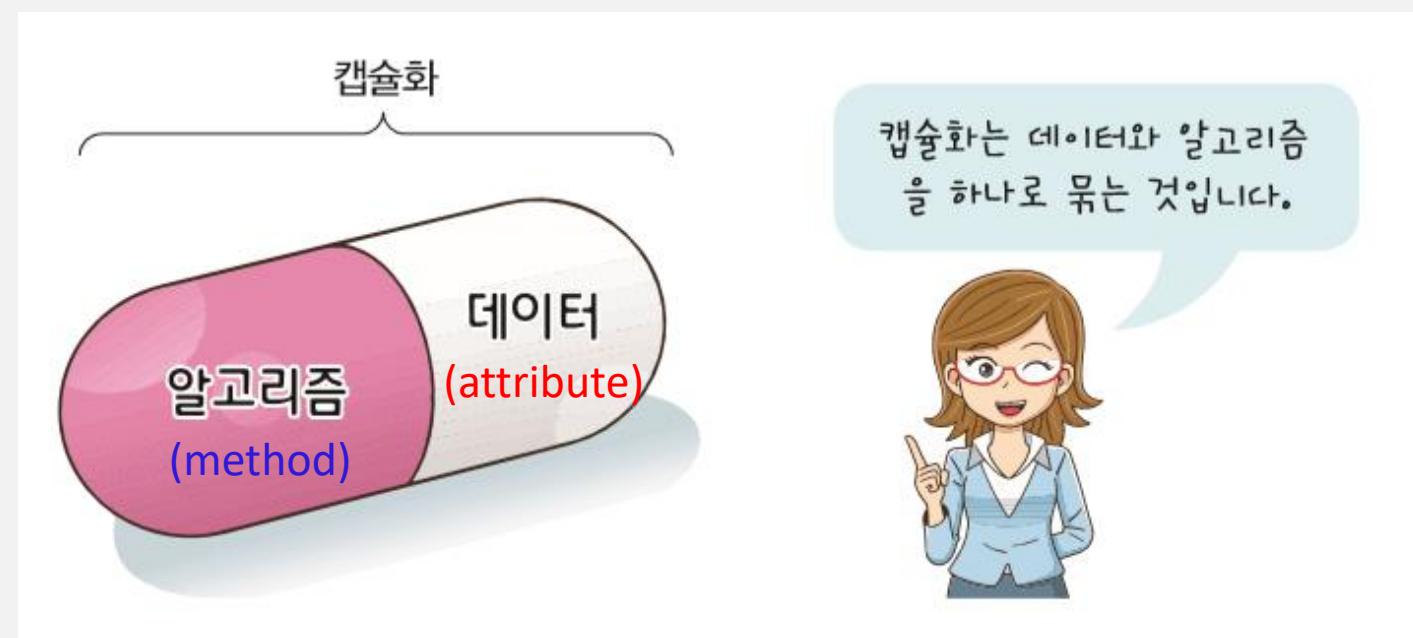


Class

- 객체 지향 프로그래밍(OOP)에서 특정 객체를 생성하기 위해 변수와 메소드를 정의하는 일종의 틀
 - Attribute(data) 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 id, member variable, member method 등 3 가지 부분으로 구성

```
class Ball:
```

← Class id

```
    size = 0
```

← Member Variable

```
    direction = "default"
```

```
def bounce(self):
```

← Member method

```
    if self.direction == "down":
```

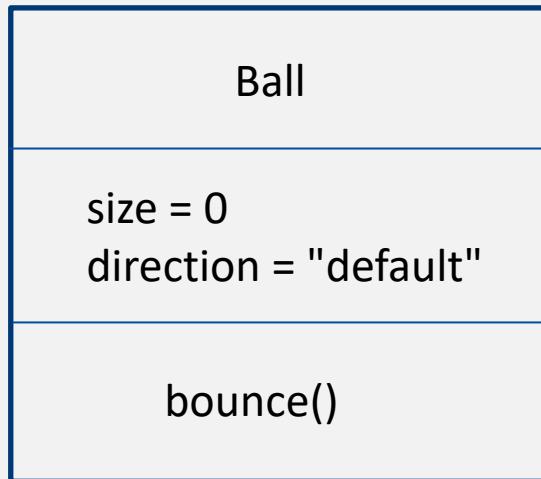
```
        self.direction = "up"
```

self는 Python 만의 독특한 변수로 class 내에서 정의되는 함수(method)는 무조건 첫번째 인자로 self를 사용하여야 함 → 후에 설명

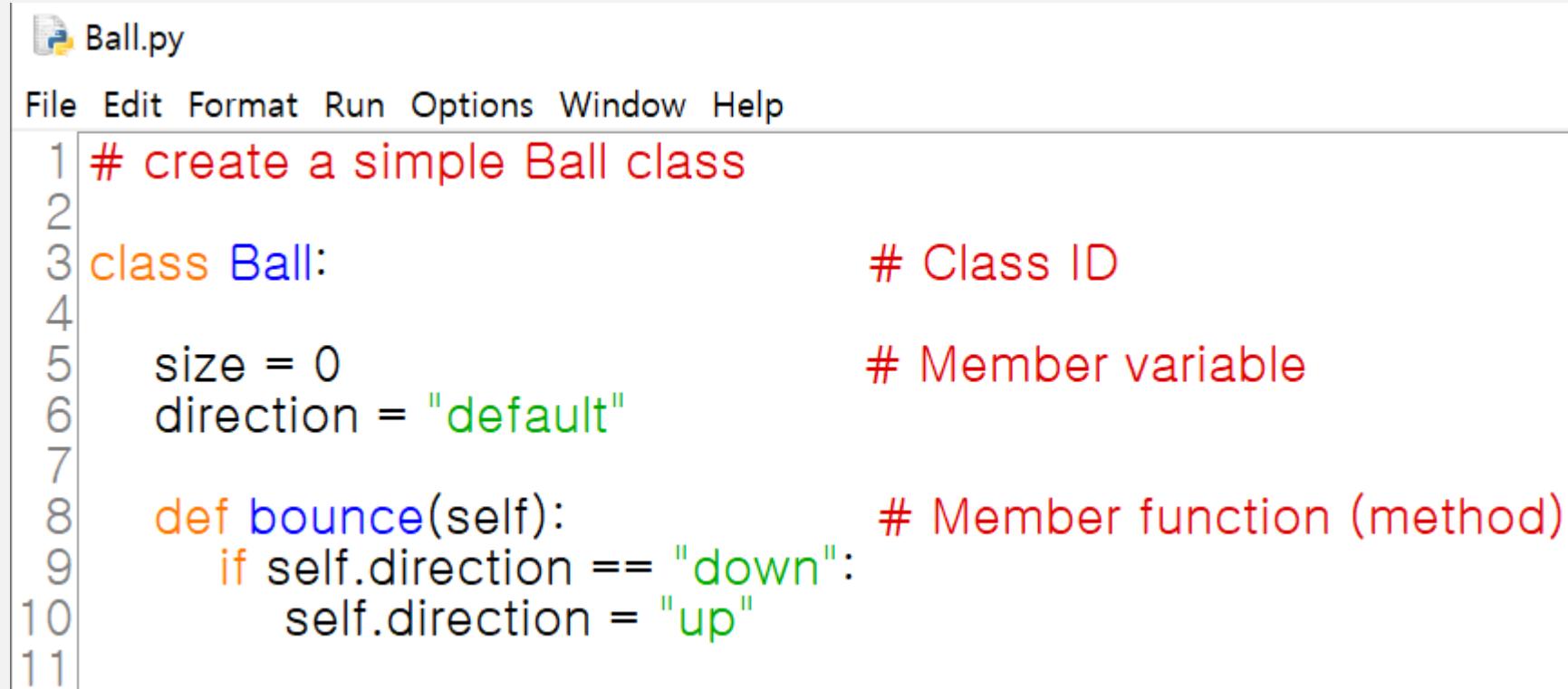
Class

- 간단한 Ball class

Id



Class notation in UML



The image shows a code editor window titled 'Ball.py'. The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code is numbered from 1 to 11. A vertical line on the left indicates the current line of code.

```
Ball.py
File Edit Format Run Options Window Help
1 # create a simple Ball class
2
3 class Ball: # Class ID
4
5     size = 0 # Member variable
6     direction = "default"
7
8     def bounce(self): # Member function (method)
9         if self.direction == "down":
10            self.direction = "up"
11
```

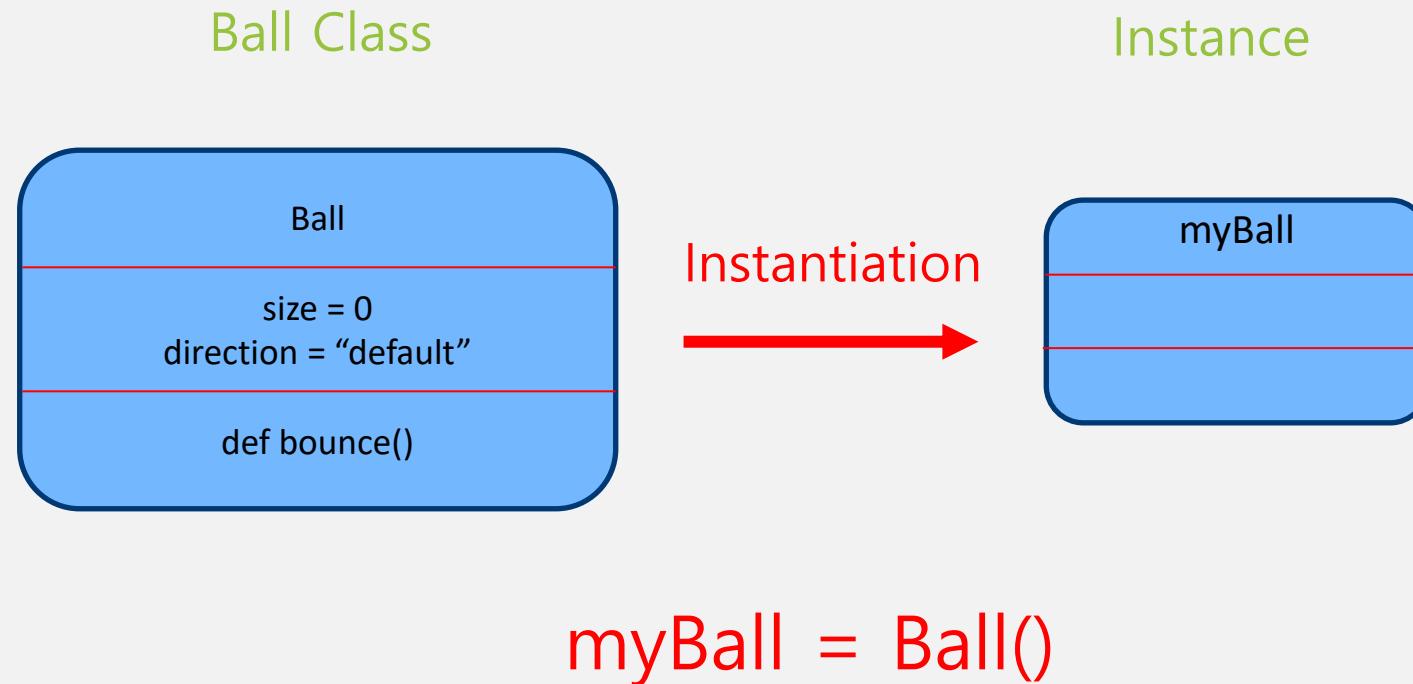
- UML : Unified Modeling Language

Instance (Object)

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



Instance(Object) 생성



Instance 생성

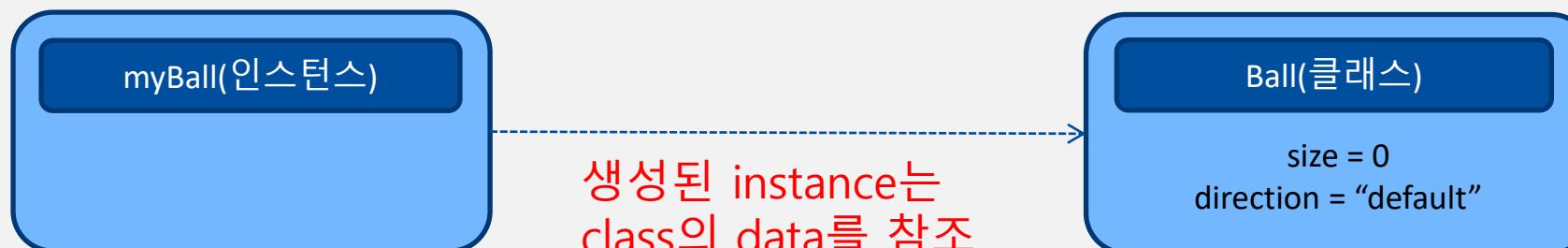
- 기본적으로 instance는 생성이 완료된 직후 원본 class와 동일한 데이터(attribute)와 함수(method)를 가짐.

```
class Ball:  
    size = 0  
    direction = "default"  
  
    def bounce(self):  
        if self.direction == "down":  
            self.direction = "up"
```

간단한 Ball class 생성

```
myBall = Ball()
```

Class의 instance인 myBall생성



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

myBall(인스턴스)

Ball(클래스)

size = 0
direction = "default"

생성된 instance는
class의 data를 참조

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 이름공간(name space)에 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
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
16 print("1) myBall.size = ", myBall.size)
17 print("2) myBall.direction = ", myBall.direction)
18
```

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

myBall(인스턴스)

direction = "down"

Ball(클래스)

size = 0
direction = "default"

Instance에 특화된 data는 instance의 이름공간(name space)에 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

myBall(인스턴스)

direction = "down" → "up"

Ball(클래스)

size = 0
direction = "default"

Instance에 특화된 data는 instance의 이름공간에 저장

```

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
16 print("1) myBall.size = ", myBall.size)
17 print("2) myBall.direction = ", myBall.direction)
18
19 myBall.bounce()
20 print("Wn3) myBall.size = ", myBall.size)
21 print("4) myBall.direction = ", myBall.direction)

```

- 1) myBall.size = 0
- 2) myBall.direction = down
- 3) myBall.size = 0
- 4) myBall.direction = up

myBall(인스턴스)

direction = "down" → "up"

Ball(클래스)

size = 0
direction = "default"

Instance에 특화된 data는 instance의 이름공간에 저장

Instance 멤버 변수 동적 추가

- Class와 Instance에 동적으로 member variable 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

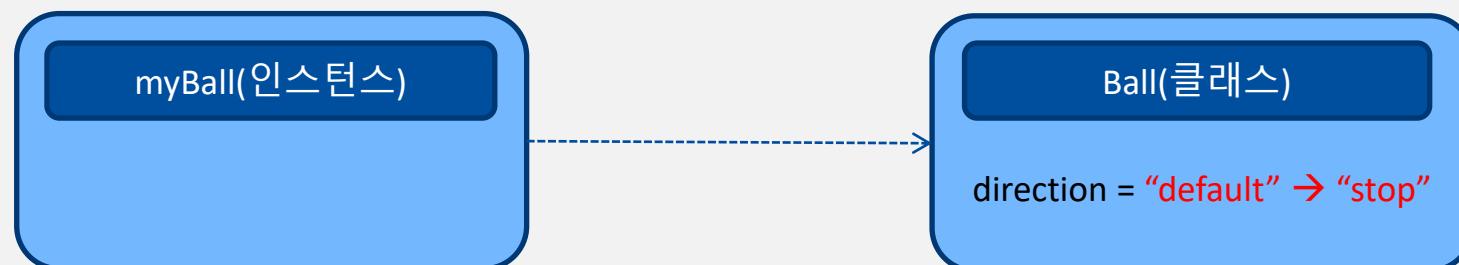


Instance에 동적으로 추가된 변수 "color"

Instance를 통한 Class 참조

- 인스턴스가 자신을 생성한 Class를 참조하기 위해 instance의 내장 속성 '__class__'를 사용 (Built-In Class Attributes in Python)

```
class Ball:  
    direction = "default"  
  
    def bounce(self):  
        if self.direction == "down":  
            self.direction = "up"  
  
myBall = Ball()  
myBall.__class__.direction = "stop" ← ' __class__ ' 속성을 이용해  
                                      class 데이터를 변경
```



생성된 instance를 통해 class 변수 값 변경

```

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("\n4) myBall.size = ", myBall.size)
23 print("5) myBall.direction = ", myBall.direction)
24
25 print("6) Class Ball의 direction 값 = ", myBall.__class__.direction)
26
27 myBall.__class__.direction = "side"
28 print("7) Class Ball의 direction 값 = ", myBall.__class__.direction)
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
 ``

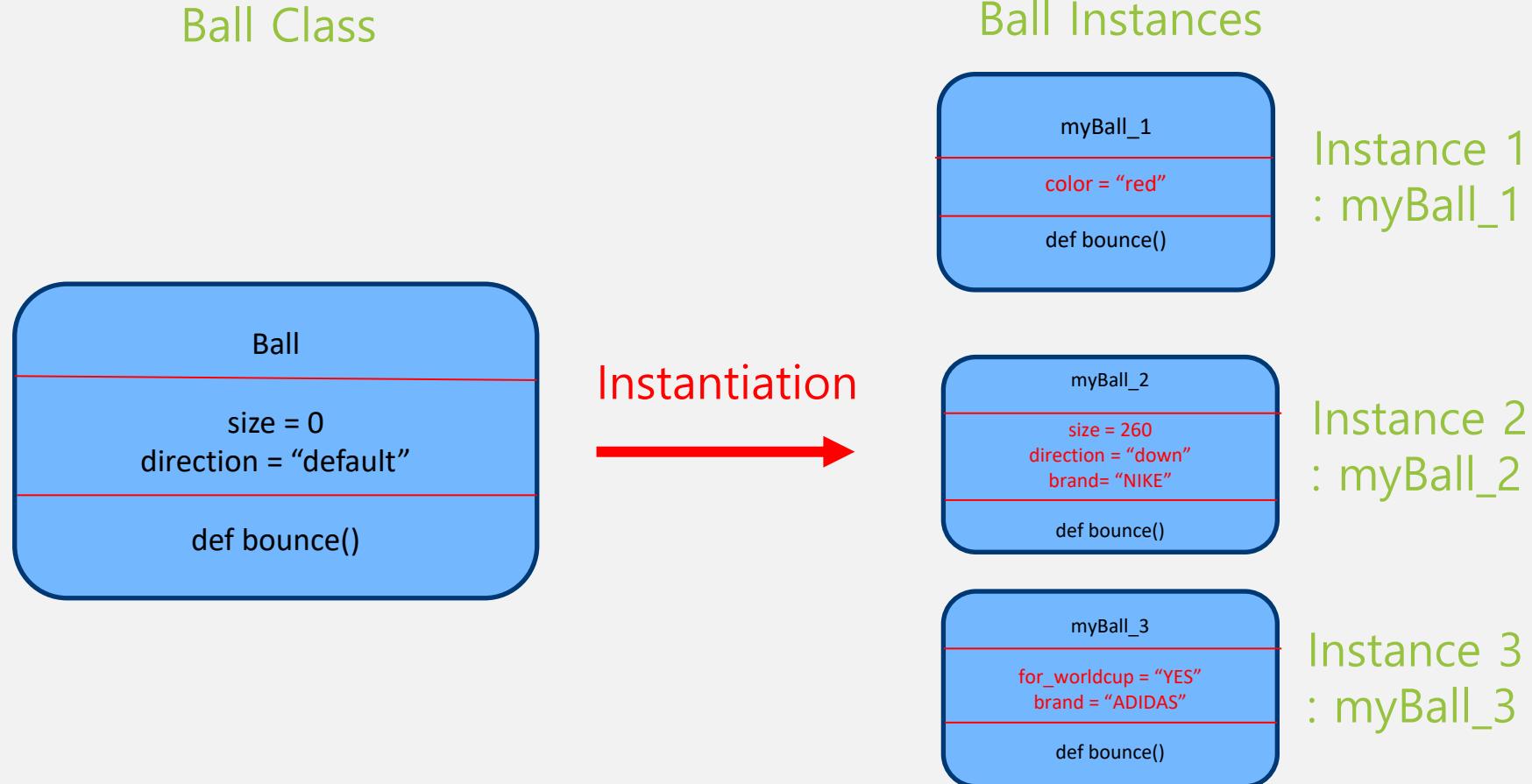
myBall(인스턴스)

direction = "down" → "up"
color = "red"

Ball(클래스)

size = 0
direction = "default" →
"side"

Instance 생성

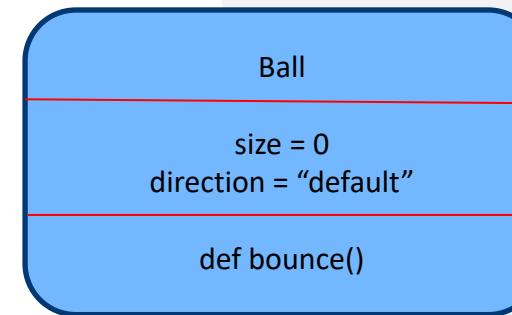


```

1 # using the Ball class
2
3 class Ball:
4
5     size = 0
6     direction = "default"
7
8     def bounce(self):
9         if self.direction == "down":
10            self.direction = "up"
11
12 #main
13 myBall_1 = Ball() # myBall_1 instance 생성
14 myBall_1.color = "red"
15
16 myBall_2 = Ball() # myBall_2 instance 생성
17 myBall_2.size = 260
18 myBall_2.direction = "down"
19 myBall_2.brand = "NIKE"
20
21 myBall_3 = Ball() # myBall_3 instance 생성
22 myBall_3.for_worldcup = "YES"
23 myBall_3.brand = "ADIDAS"

```

Ball Class



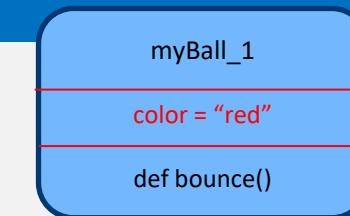
Ball

size = 0
direction = "default"
def bounce()

Instantiation



Ball Instances

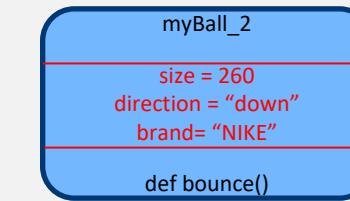


myBall_1

color = "red"

def bounce()

Instance 1
: myBall_1



myBall_2

size = 260

direction = "down"

brand = "NIKE"

def bounce()

Instance 2
: myBall_2



myBall_3

for_worldcup = "YES"

brand = "ADIDAS"

def bounce()

Instance 3
: myBall_3

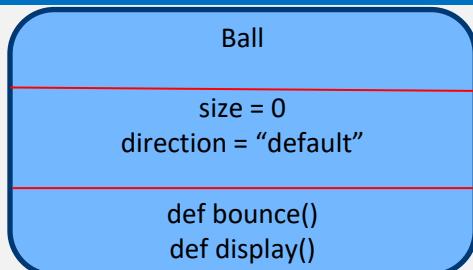
```

1 D = True
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     def display(self):
12         print("d1) self =", self)
13         print("d2) self.size =", self.size)
14         print("d3) self.direction =", self.direction)
15
16 #main
17 #myBall_1 관련
18 myBall_1 = Ball() # myBall_1 instance 생성
19 if D:
20     print("m1) myBall_1 =", myBall_1)
21
22 myBall_1.color = "red"
23 myBall_1.display()
24 if D:
25     print("m1-1) myBall_1.color =", myBall_1.color)
26
27
28 #myBall_2 관련
29 myBall_2 = Ball() # myBall_2 instance 생성
30 if D:
31     print("Wnm2) myBall_2 =", myBall_2)
32
33 myBall_2.size = 260
34 myBall_2.direction = "down"
35 myBall_2.brand = "NIKE"
36 myBall_2.display()
37 if D:
38     print("m2-1) myBall_2.brand =", myBall_2.brand)
39
40 #myBall_3 관련
41 myBall_3 = Ball() # myBall_3 instance 생성
42 if D:
43     print("Wnm3) myBall_3 =", myBall_3)
44
45 myBall_3.for_worldcup = "YES"
46 myBall_3.brand = "ADIDAS"
47 myBall_3.display()
48 if D:
49     print("m3-1) myBall_3.for_worldcup =", myBall_3.for_worldcup)
50     print("m3-2) myBall_3.brand =", myBall_3.brand)

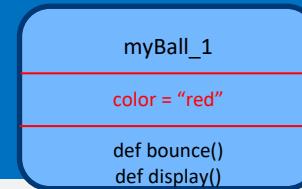
```

Ball Instances

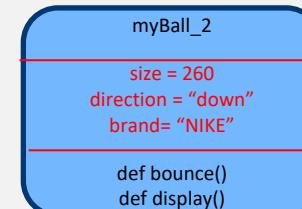
Ball Class



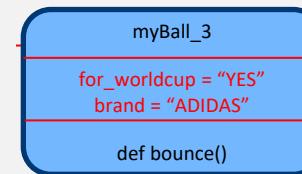
Instantiation



Instance 1 : myBall_1



Instance 2 : myBall_2

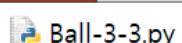


Instance 3 : myBall_3

m1) myBall_1 = <__main__.Ball object at 0x000001DC303D8910>
d1) self = <__main__.Ball object at 0x000001DC303D8910>
d2) self.size = 0
d3) self.direction = default
m1-1) myBall_1.color = red

m2) myBall_2 = <__main__.Ball object at 0x000001DC30466FA0>
d1) self = <__main__.Ball object at 0x000001DC30466FA0>
d2) self.size = 260
d3) self.direction = down
m2-1) myBall_2.brand = NIKE

m3) myBall_3 = <__main__.Ball object at 0x000001DC30466BE0>
d1) self = <__main__.Ball object at 0x000001DC30466BE0>
d2) self.size = 0
d3) self.direction = default
m3-1) myBall_3.for_worldcup = YES
m3-2) myBall_3.brand = ADIDAS



```
File Edit Format Run Options Window Help
1 D = True
2
3 class Ball:
4
5     size = 0
6     direction = "default"
7
8     def bounce(self):
9         if self.direction == "down":
10            self.direction = "up"
11
12     def display(self):
13         print("d1) self =", self)
14         print("d2) self.size =", self.size)
15         print("d3) self.direction =", self.direction)
16
```

```
m1) myBall_1 = <__main__.Ball object at 0x000001D993CF69D0>
d1) self = <__main__.Ball object at 0x000001D993CF69D0>
d2) self.size = 0
d3) self.direction = default
m1-1) myBall_1.color = red

m2) myBall_2 = <__main__.Ball object at 0x000001D993CF6BB0>
d1) self = <__main__.Ball object at 0x000001D993CF6BB0>
d2) self.size = 260
d3) self.direction = down
m2-1) myBall_2.brand = NIKE

m3) myBall_3 = <__main__.Ball object at 0x000001D993CF6B20>
d1) self = <__main__.Ball object at 0x000001D993CF6B20>
d2) self.size = 0
d3) self.direction = default
m3-1) myBall_3.for_worldcup = YES
m3-2) myBall_3.brand = ADIDAS
```

```
17
18 class Main: # driver class
19
20     def main(self):
21         #myBall_1 관련
22         myBall_1 = Ball() # myBall_1 instance 생성
23         if D:
24             print("m1) myBall_1 =", myBall_1)
25
26         myBall_1.color = "red"
27         myBall_1.display()
28         if D:
29             print("m1-1) myBall_1.color =", myBall_1.color)
30
31         #myBall_2 관련
32         myBall_2 = Ball() # myBall_2 instance 생성
33         if D:
34             print("Wnm2) myBall_2 =", myBall_2)
35
36         myBall_2.size = 260
37         myBall_2.direction = "down"
38         myBall_2.brand = "NIKE"
39         myBall_2.display()
40         if D:
41             print("m2-1) myBall_2.brand =", myBall_2.brand)
42
43         #myBall_3 관련
44         myBall_3 = Ball() # myBall_3 instance 생성
45         if D:
46             print("Wnm3) myBall_3 =", myBall_3)
47
48         myBall_3.for_worldcup = "YES"
49         myBall_3.brand = "ADIDAS"
50         myBall_3.display()
51         if D:
52             print("m3-1) myBall_3.for_worldcup =", myBall_3.for_worldcup)
53             print("m3-2) myBall_3.brand =", myBall_3.brand)
54
55     #main
56     m = Main()
57     m.main()
```

Class 형식

Class Class-ID:
attribute list

def method_1(self, argument list):

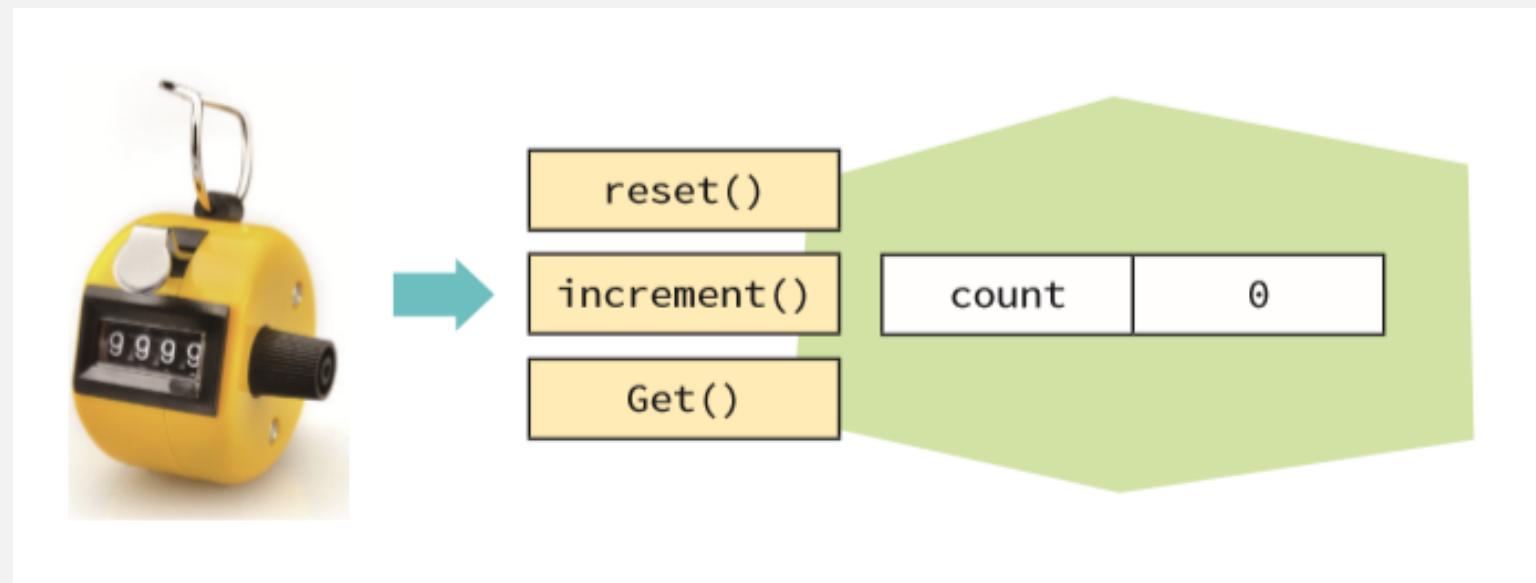
....

def method_2(self, argument list):

....

Class의 예

- Counter Class를 작성하여 보자. Counter Class는 기계식 **계수기**를 나타내며 경기장이나 콘서트에 입장하는 **관객 수**를 세기 위하여 사용할 수 있음.
- 버튼을 누를 때마다 입장하는 관객수가 1명씩 **증가**함.
- 사용완료후 0으로 **리셋**하는 기능이 있음.
- 현재의 관객수를 **확인**할 수 있는 기능이 필요함.



```

1 class Counter:
2
3     count = 0
4
5     def reset(self):
6         self.count = 0
7
8     def increment(self):
9         self.count += 1
10
11    def get(self):
12        return self.count
13
14 def counter_operate(counter_obj, how_many_click):
15     counter_obj.reset()
16     print("Wnc1) counter_obj의 count 초기값 :", counter_obj.get())
17
18     for i in range(1, how_many_click+1):
19         counter_obj.increment()
20         print(">> {}번째 click 후, counter_obj의 값 : {}".format(i, counter_obj.get()))
21
22     counter_obj.reset()
23     print("Wnc2) reset 버튼 click 후 counter_obj 값 :", counter_obj.get())
24
25
26 # main
27 counter1 = Counter()
28 counter2 = Counter()
29
30 n = int(input(">> counter1 click 횟수를 입력하시오 : "))
31
32 print("1) counter1을 {}번 click".format(n))
33 counter_operate(counter1, n)
34
35 m = int(input("Wn>> counter2 click 횟수를 입력하시오 : "))
36
37 print("2) counter2를 {}번 click".format(m))
38 counter_operate(counter2, m)
39

```

>> counter1 click 횟수를 입력하시오 : 4
1) counter1을 4번 click

c1) counter_obj의 count 초기값 : 0
>> 1번째 click 후, counter_obj의 값 : 1
>> 2번째 click 후, counter_obj의 값 : 2
>> 3번째 click 후, counter_obj의 값 : 3
>> 4번째 click 후, counter_obj의 값 : 4

c2) reset 버튼 click 후 counter_obj 값 : 0

>> counter2 click 횟수를 입력하시오 : 3
2) counter2를 3번 click

c1) counter_obj의 count 초기값 : 0
>> 1번째 click 후, counter_obj의 값 : 1
>> 2번째 click 후, counter_obj의 값 : 2
>> 3번째 click 후, counter_obj의 값 : 3

c2) reset 버튼 click 후 counter_obj 값 : 0

객체 초기화 - 생성자(constructor) method

- Class로 부터 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.49

```
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 설정.
- 하지만 이 기능 만이 생성자의 기능이 아님.
- Constructor argument

생성자를 통해 instance 생성 시 초기화 할 멤버변수 값을 argument로 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

```

1 class Ball:
2     def __init__(self, color, size, direction): # Constructor(생성자)
3         print ("Wnc1) 생성자")                  # __init__() method
4         self.color = color
5         self.size = size
6         self.direction = direction
7
8         print ("c2) self.color = ",color)
9         print ("c3) self.size = ",size)
10        print ("c4) self.direction = ",direction)
11
12    def bounce(self):
13        print ("Wnb1) in bounce()")
14        if self.direction == "down":
15            self.direction = "up"
16
17 #main
18 print ("1) main")
19 myBall = Ball("red", "small", "down") # Create an instance
20                                # with 3 attributes
21 print ("Wn2) myBall.size : ", myBall.size)
22 print ("3) myBall.color : ", myBall.color)
23 print ("4) myBall.direction : ", myBall.direction)
24
25 myBall.bounce()
26 print ("Wn5) myBall.direction : ", myBall.direction)
27

```

1) main

c1) 생성자

c2) self.color = red

c3) self.size = small

c4) self.direction = down

2) myBall.size : small

3) myBall.color : red

4) myBall.direction : down

b1) in bounce()

5) myBall.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 self.count = 0
5 print("ini) self = {}. self.count = {}".format(self, self.count))
6
7 def reset(self):
8 self.count = 0
9 print("res) self = {}. self.count = {}".format(self, self.count))
10
11 def increment(self):
12 self.count += 1
13 print("inc) self = {}. self.count = {}".format(self, self.count))
14
15 def get(self):
16 print("get) self = {}. self.count = {}".format(self, self.count))
17 return self.count
18
19 #main
20 print("1) Create counter1.")
21 counter1 = Counter()
22 print(">> counter1 = ", counter1)
23
24 print("Wn2) Create counter2.")
25 counter2 = Counter()
26 print(">> counter2 = ", counter2, "Wn")
27
28 counter1.reset()
29 print("Wn3) counter1.count =", counter1.get())
30
31 counter1.increment()
32 print("Wn4) counter1.count =", counter1.get())
33
34 counter2.reset()
35 print("Wn5) counter2.count =", counter2.get())
36
37 counter2.increment()
38 counter2.increment()
39 print("Wn6) counter2.count =", counter2.get())
40
```

1) Create counter1.  
ini) self = <\_\_main\_\_.Counter object at 0x0000026056A969D0>. self.count = 0  
>> counter1 = <\_\_main\_\_.Counter object at 0x0000026056A969D0>  
  
2) Create counter2.  
ini) self = <\_\_main\_\_.Counter object at 0x0000026056A96FD0>. self.count = 0  
>> counter2 = <\_\_main\_\_.Counter object at 0x0000026056A96FD0>  
  
res) self = <\_\_main\_\_.Counter object at 0x0000026056A969D0>. self.count = 0  
get) self = <\_\_main\_\_.Counter object at 0x0000026056A969D0>. self.count = 0  
  
3) counter1.count = 0  
inc) self = <\_\_main\_\_.Counter object at 0x0000026056A969D0>. self.count = 1  
get) self = <\_\_main\_\_.Counter object at 0x0000026056A969D0>. self.count = 1  
  
4) counter1.count = 1  
res) self = <\_\_main\_\_.Counter object at 0x0000026056A96FD0>. self.count = 0  
get) self = <\_\_main\_\_.Counter object at 0x0000026056A96FD0>. self.count = 0  
  
5) counter2.count = 0  
inc) self = <\_\_main\_\_.Counter object at 0x0000026056A96FD0>. self.count = 1  
inc) self = <\_\_main\_\_.Counter object at 0x0000026056A96FD0>. self.count = 2  
get) self = <\_\_main\_\_.Counter object at 0x0000026056A96FD0>. self.count = 2  
  
6) counter2.count = 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()`

sys — System-specific parameters

docs.python.org/3/library/sys.html

getrefcount 1/2

- On Unix, the encoding is the locale encoding.
- On Windows, the encoding may be 'utf-8' or 'mbcs', depending on user configuration.
- On Android, the encoding is 'utf-8'.
- On VxWorks, the encoding is 'utf-8'.

*Changed in version 3.2:* `getfilesystemencoding()` result cannot be `None` anymore.

*Changed in version 3.6:* Windows is no longer guaranteed to return 'mbcs'. See [PEP 529](#) and `_enablelegacywindowsfsencoding()` for more information.

*Changed in version 3.7:* Return 'utf-8' in the UTF-8 mode.

### `sys.getfilesystemencodeerrors()`

Return the name of the error mode used to convert between Unicode filenames and bytes filenames. The encoding name is returned from `getfilesystemencoding()`.

`os.fsencode()` and `os.fsdecode()` should be used to ensure that the correct encoding and errors mode are used.

*New in version 3.6.*

### `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()`.

### `sys.getrecursionlimit()`

Return the current value of the recursion limit, the maximum depth of the Python interpreter stack. This limit prevents infinite recursion from causing an overflow of the C stack and crashing Python. It can be set by `setrecursionlimit()`.

### `sys.getsizeof(object[, default])`

Return the size of an object in bytes. The object can be any type of object. All built-in objects will return correct results, but this does not have to hold true for third-party extensions as it is implementation specific.

Only the memory consumption directly attributed to the object is accounted for, not the memory consumption of objects it refers to.

If given, `default` will be returned if the object does not provide means to retrieve the size. Otherwise a `TypeError` will be raised.

`getsizeof()` calls the object's `__sizeof__` method and adds an additional garbage collector overhead if the object is managed by the garbage collector.

```

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 = "down"
11
12 def __init__(self, color, size, direction): # 생성자(constructor)
13 print("Wn1-1) 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("Wn1-2) 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 때마다 reference count가 1 감소
34 # 카운트가 0 이 되면 객체는 메모리 공간에서 삭제
35
36 print("1) main ")
37 print("Wn2) myBall 객체 생성")
38 myBall = Ball("red", "small", "down") # myBall instance 생성
39 # reference counter 증가 : 1
40 print("Wn3) 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 = down

## 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:\W과소사\W과소사-강의예제\ Destructor.py", line 78, in <module>  
print("17) myBall의 reference count = ", sys.getrefcount(myBall))

NameError: name 'myBall' is not defined

```

45 print("Wn5) call myBall.bounce()")
46 myBall.bounce()
47 print("Wn6) myBall.direction = ", myBall.direction)
48 print("7) myBall의 reference count = ", sys.getrefcount(myBall))
49
50 # reference count 증가 예
51 yourBall = myBall
52 print("Wn8) myBall의 reference count = ", sys.getrefcount(myBall))
53
54 herBall = myBall
55 print("9) myBall의 reference count = ", sys.getrefcount(myBall))
56
57 del yourBall
58 print("Wn10) myBall의 reference count = ", sys.getrefcount(myBall))
59
60 del herBall
61 print("11) myBall의 reference count = ", sys.getrefcount(myBall))
62
63 # myBall 객체에 대한 참조를 모두 해제
64 print("12) myBall 객체에 대한 참조를 모두 해제")
65
66 del myBall.size, myBall.color, myBall.direction
67 print("13) myBall의 reference count = ", sys.getrefcount(myBall))
68
69 print("14) myBall의 size, color, direction 값들이 class member variable 초기값으로 초기화")
70 print(" myBall.size = ", myBall.size)
71 print(" myBall.color = ", myBall.color)
72 print(" myBall.direction = ", myBall.direction)
73 print("15) myBall의 reference count = ", sys.getrefcount(myBall))
74
75 del myBall
76 print("16) 소멸자가 자동으로 불리어져서 myBall 객체가 메모리에서 사라짐.")
77 print("17) myBall의 reference count = ", sys.getrefcount(myBall))
78
79

```

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 = down  
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:\W소사\W강의예제\Destructor.py", line 78, in <module>  
print("17) myBall의 reference count = ", sys.getrefcount(myBall))  
NameError: name 'myBall' is not defined. Did you mean: 'Ball'?

# Method 정의

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

# self

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

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

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

- 위의 예에서 bounce()라는 method 입장에서는 어느 instance가 자신을 호출했는지 알아야 함
- self 인자는 어느 객체가 method를 호출했는지 알려줌  
→ 이를 **instance reference**라 함.

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

```

1 class Ball:
2 size = 0
3 direction = "default"
4
5 def __init__(self, in_size, in_direction):
6 print("i-1) In init, self =", self)
7 self.size = in_size
8 self.direction = in_direction
9
10 print("i-2) self.size =", self.size)
11 print("i-3) self.dirction =", self.direction, "\n")
12
13 def bounce(self):
14 print("b-1) self =", self)
15 if self.direction == "down":
16 self.direction = "up"
17
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 = &lt;\_\_main\_\_.Ball object at 0x000001D8E8EC6970&gt;

i-2) self.size = 10

i-3) self.dirction = down

3) yourBall 생성

i-1) In init, self = &lt;\_\_main\_\_.Ball object at 0x000001D8E8EC6C40&gt;

i-2) self.size = 20

i-3) self.dirction = up

4) call myBall.bounce()

b-1) self = &lt;\_\_main\_\_.Ball object at 0x000001D8E8EC6970&gt;

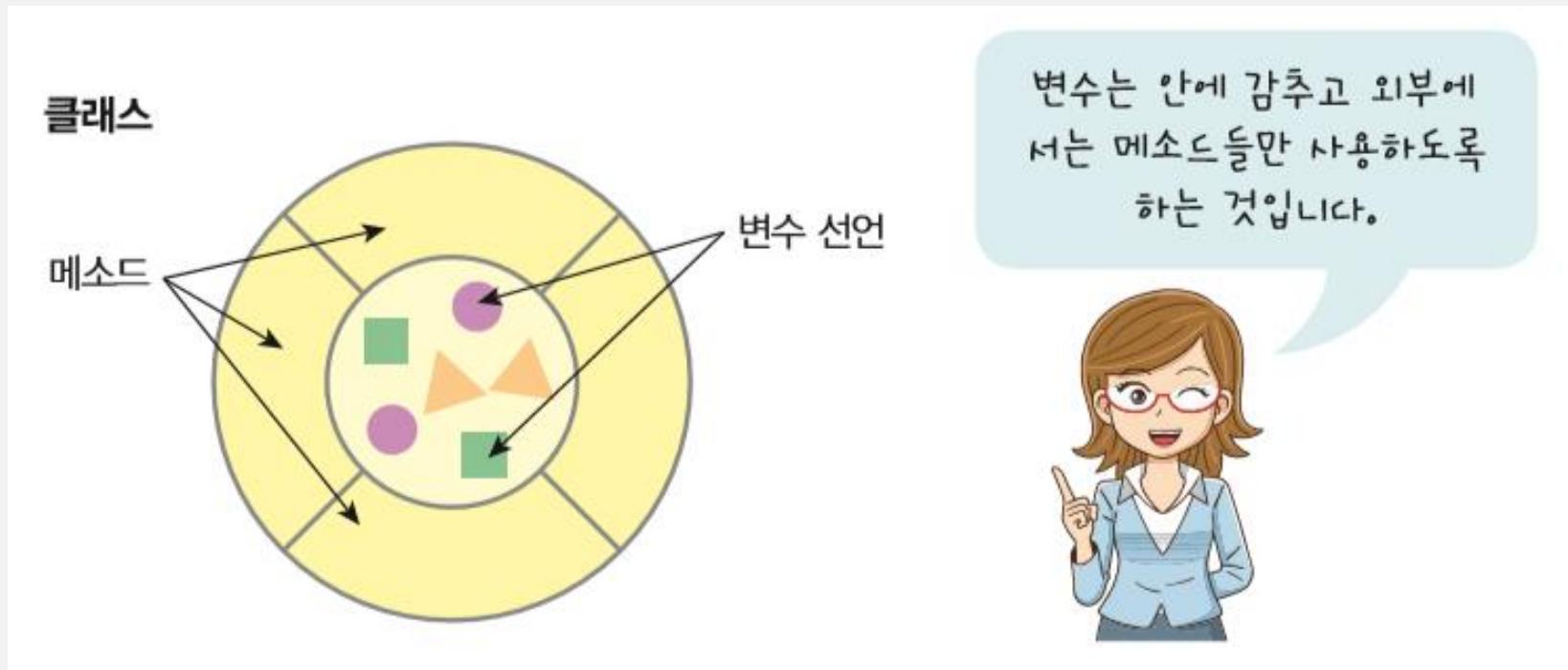
5) call yourBall.bounce()

b-1) self = &lt;\_\_main\_\_.Ball object at 0x000001D8E8EC6C40&gt;

&gt;&gt;&gt;

# 정보은닉(Information Hiding)

- 구현의 세부 사항을 Class 안에 감추는 것



# Private instance variable

- Instance variable을 private으로 선언하여 사용하는 것을 권장
  - 선언하고자 하는 변수 앞에 \_ (underline 2개)를 붙이면 private으로 선언됨
- Method도 이름 앞에 \_를 붙이면 private method가 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'

File Edit Format Run Options Window Help

```

1 class Student:
2 def __init__(self, name=None, age=0):
3 print("i-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 s1 생성")
13 s1 = Student("Park", 20)
14
15 print("i-3) private 선언된 __age 값을 직접 접근시 error 발생")
16 print(s1.__age)
17

```

1) main start  
2) 객체 s1 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:\소사\강의예제\student.py", line 16, in <module>  
 print(s1.\_\_age)

AttributeError: 'Student' object has no attribute '\_\_age'

# 접근자와 설정자

- Private 선언된 변수(attribute)를 읽어오고 변수값을 변경하는 방법  
→ 접근자와 설정자에게 위탁(delegation)
  - 접근자(getter, accessor) : instance 변수값 (attributes)을 반환 (**read only**)
  - 설정자(setter, mutator) : instance 변수값(attributes)을 설정(변경) (**write**)



```

File Edit Format Run Options Window Help
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 getName(self): # getter for __name
10 print("ngetName) self.__name =", self.__name)
11 return self.__name
12
13 def getAge(self): #getter for __age
14 print("ngetAge) self.__age =", self.__age)
15 return self.__age
16
17 def setName(self, name): # setter for __name
18 self.__name=name
19 print("nsetName) self.__name =", self.__name)
20
21 def setAge(self, age): # setter for __age
22 self.__age=age
23 print("setAge) self.__age =", self.__age)
24
#main
26 print("1) main start")
27 print("2) 객체 s1 생성")
28 s1 = Student("Park", 20)
29
30 # error !!
31 #print(s1.__age)
32
33 print("3) s1.__name =", s1.getName())
34 print("4) s1.__age =", s1.getAge())
35
36 print("n5) s1 attribute __name, __age 값 변경")
37 s1.setName("Hyun")
38 s1.setAge(25)
39
40 print("6) s1.__name =", s1.getName())
41 print("7) s1.__age =", s1.getAge())
42

```

1) main start  
 2) 객체 s1 생성  
 i-1) name과 age를 private 선언  
 i-2) self.\_\_name = Park  
 self.\_\_age = 20  
 getName) self.\_\_name = Park  
 3) s1.\_\_name = Park  
 getAge) self.\_\_age = 20  
 4) s1.\_\_age = 20  
 5) s1 attribute \_\_name, \_\_age 값 변경  
 setName) self.\_\_name = Hyun  
 setAge) self.\_\_age = 25  
 getName) self.\_\_name = Hyun  
 6) s1.\_\_name = Hyun  
 getAge) self.\_\_age = 25  
 7) s1.\_\_age = 25

```

1 class Television:
2
3 channel = 0
4 volume = 0
5 on = "off"
6
7 def __init__(self, channel, volume, on):
8 print("init-1) In init")
9 self.channel = channel
10 self.volume = volume
11 self.on = on
12 print("init-2) self.channel =", self.channel)
13 print(" self.volume =", self.volume)
14 print(" self.on =", self.on)
15
16 # getters
17 def getChannel(self):
18 print("getChannel) self.channel =", self.channel)
19 return self.channel
20
21 def getVolume(self):
22 print("getVolume) self.volume =", self.volume)
23 return self.volume
24
25 def getOn(self):
26 print("getOn) self.On =", self.on)
27 return self.on
28
29 #setters
30 def setChannel(self, channel):
31 self.channel = channel
32 print("setChannel) self.channel =", self.channel)
33
34 def setVolume(self, volume):
35 self.volume = volume
36 print("setVolumw) self.volume =", self.volume)
37
38 def setOn(self, on):
39 self.on = on
40 print("setOn) self.On =", self.on)
41
42 # show all current values of atttibutes
43 def show(self):
44 print("show) self.channel =", self.channel)
45 print(" self.volume =", self.volume)
46 print(" self.on =", self.on)
47
48

```

```

49 class Main: # driver class
50
51 def main(self):
52
53 print("1) main start")
54 print("Wn2) 객체 t 생성")
55 t = Television(9, 10, True)
56 t.show()
57
58 print("Wn3) 채널변경")
59 t.setChannel(11)
60 t.getChannel()
61
62 print("Wn4) 볼륨 변경")
63 t.setVolume(20)
64 t.getVolume()
65
66 print("Wn5) 전원끄기")
67 t.setOn(False)
68 t.getOn()
69
70 # triggering
71 m = Main()
72 m.main()
73

```

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

# 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("Circleinit-1) 반지름 값이 0 보다 작습니다.")
9 exit()
10 else:
11 self.__radius = radius
12 self.__Pie = math.pi
13 print("Circleinit-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, rad = 1.0):
33 self.__input_radius = rad
34 print("Ra_init) self.__input_radius =", self.__input_radius)
35
36 def in_value(self):
37 self.__r = input("Wn>> 원의 반지름을 입력하시오. 기본값 (1) : ")
38
39 if self.__r == "":
40 print("Ra_in_value-1) default self.__input_radius =", self.__input_radius)
41
42 else:
43 self.__input_radius = float(self.__r)
44 print("Ra_in_value-2) self.__input_radius =", self.__input_radius)
45
46 return self.__input_radius
47
48 class Main: # driver
49
50 __r = 0
51
52 def main(self):
53 print("1) main")
54 rad = Radius_input()
55
56 self.__r = rad.in_value()
57 print("Wn2) self.__r =", self.__r)
58
59 c1 = Circle(self.__r)
60
61 print("Wn3) 원의 반지름", c1.getRadius())
62 print("4) 원의 넓이=", c1.calcArea())
63 print("5) 원의 둘레=", c1.calcCircum())
64
65 print("6) setRadius() 이용")
66 c1.setRadius(10)
67 print("Wn7) 원의 반지름=", c1.getRadius())
68 print("8) 원의 넓이=", c1.calcArea())
69 print("9) 원의 둘레=", c1.calcCircum())
70
71 # Ignition
72 m = Main()
73 m.main()

```

1) main  
Ra\_init) self.\_\_input\_radius = 1.0  
>> 원의 반지름을 입력하시오. 기본값 (1) : Ra\_in\_value-1) default self.\_\_input\_radius = 1.0  
2) self.\_\_r = 1.0  
Circleinit-2) self.\_\_radius = 1.0  
 self.\_\_Pie = 3.141592653589793  
3) 원의 반지름 1.0  
4) 원의 넓이= 3.141592653589793  
5) 원의 둘레= 6.283185307179586  
6) setRadius() 이용  
7) 원의 반지름= 10  
8) 원의 넓이= 314.1592653589793  
9) 원의 둘레= 62.83185307179586

```

1 import math
2
3 class Circle:
4
5 def __init__(self, radius):
6
7 if radius <= 0:
8 print("Circleinit-1) 반지름 값이 0 보다 작습니다.")
9 exit()
10 else:
11 self.__radius = radius
12 self.__Pie = math.pi
13 print("Circleinit-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
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73

```

```

class Radius_input:
31
32 def __init__(self, rad = 1.0):
33 self.__input_radius = rad
34 print("Ra_init) self.__input_radius =", self.__input_radius)
35
36 def in_value(self):
37 self.__r = input("Wn>> 원의 반지름을 입력하시오. 기본값 (1) : ")
38
39 if self.__r == "":
40 print("Ra_in_value-1) default self.__input_radius =", self.__input_radius)
41
42 else:
43 self.__input_radius = float(self.__r)
44 print("Ra_in_value-2) self.__input_radius =", self.__input_radius)
45
46 return self.__input_radius
47
48 class Main: # driver
49
50 __r = 0
51
52 def main(self):
53 print("1) main")
54 rad = Radius_input()
55
56 self.__r = rad.in_value()
57 print("Wn2) self.__r =", self.__r)
58
59 c1 = Circle(self.__r)
60
61 print("Wn3) 원의 반지름", c1.getRadius())
62 print("4) 원의 넓이=", c1.calcArea())
63 print("5) 원의 둘레=", c1.calcCircum())
64
65 print("6) setRadius() 이용")
66 c1.setRadius(10)
67 print("Wn7) 원의 반지름=", c1.getRadius())
68 print("8) 원의 넓이=", c1.calcArea())
69 print("9) 원의 둘레=", c1.calcCircum())
70
71 # Ignition
72 m = Main()
73 m.main()

```

1) main  
Ra\_init) self.\_\_input\_radius = 1.0  
>> 원의 반지름을 입력하시오. 기본값 (1) : 3  
Ra\_in\_value-2) self.\_\_input\_radius = 3.0

2) self.\_\_r = 3.0  
Circleinit-2) self.\_\_radius = 3.0  
 self.\_\_Pie = 3.141592653589793

3) 원의 반지름 3.0  
4) 원의 넓이= 28.274333882308138  
5) 원의 둘레= 18.84955592153876  
6) setRadius() 이용

7) 원의 반지름= 10  
8) 원의 넓이= 314.1592653589793  
9) 원의 둘레= 62.83185307179586

# 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

bank = BankAccount()
bank.deposit(100)
bank.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)

main
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("printArea) 반복횟수 =", n)
15 while n >= 1:
16 print(">> r.side =", r.side, "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)
```

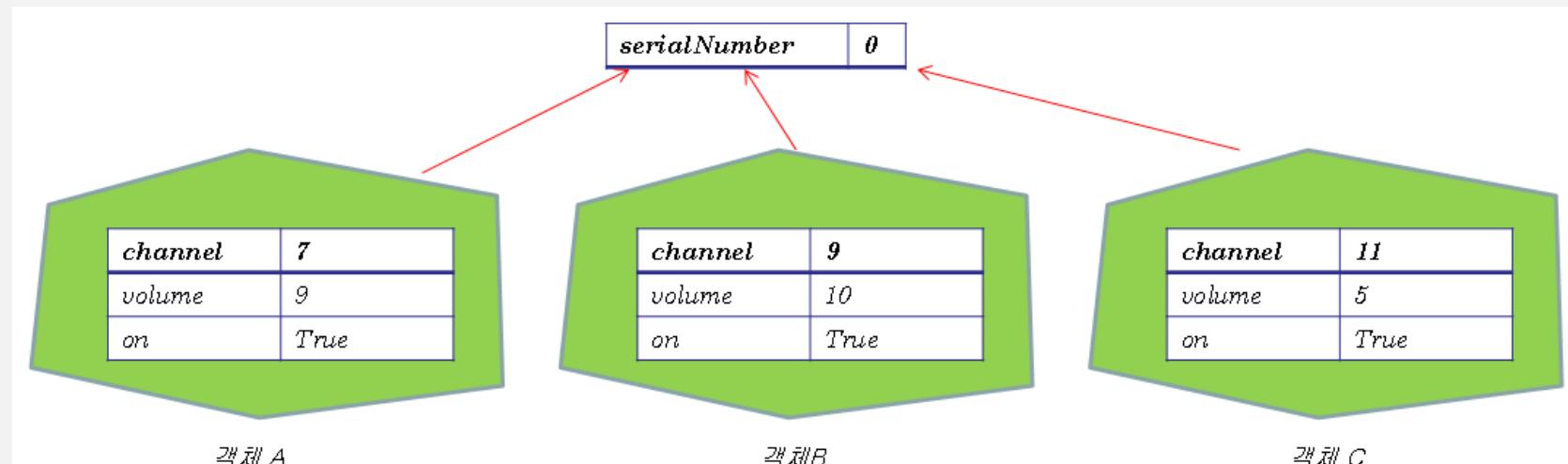
```
1) main
init) self.side = 0
printArea) 반복횟수 = 5
>> r.side = 0 area = 0
>> r.side = 1 area = 1
>> r.side = 2 area = 4
>> r.side = 3 area = 9
>> r.side = 4 area = 16
```

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

```

```

50 print("Wn11) t1.count =", t1.__getCount__())
51 print("12) t2.count =", t2.__getCount__())
52 print("13) t3.count =", t3.__getCount__())
53 print("14) t4.count =", t4.__getCount__())
54
55 print("Wn15) t1.count =", t1.count)
56 print(" t2.count =", t2.count)
57 print(" t3.count =", t3.count)
58 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 표시할지 알려줌
  - 기본출력 내용
    - Instance가 정의된 곳 (앞의 예제에서는 `_main_`)
    - Class 이름 (앞의 예제에서는 Ball)
    - Instance가 저장되어 있는 메모리상의 위치

# Default `__str__()`

- **Special method**

- class를 정의 시 Python이 자동으로 포함시키는 method들

- 그 중의 하나가 특수 method

- ‘`__str__()`’

- 객체를 `print`로 출력할 때 Python이 어떤 내용으로 표시할지 알려줌

- 기본출력 내용

- Instance가 정의된 곳 (앞의 예제에서는 `__main__`)
    - Class 이름 (앞의 예제에서는 `Ball`)
    - Instance가 저장되어 있는 메모리상의 위치

 default\_str\_.py

File Edit Format Run Options Window Help

```
1 # default __str__()를 이용하여 객체에 대한 기본정보를 출력
2
3 class Ball:
4 def __init__(self, color, size, direction):
5 print("Wninit) self =", self)
6 self.color = color
7 self.size = size
8 self.direction = direction
9
10 #main
11 myBall = Ball("red", "small", "down")
12 print (myBall)
13
14 yourBall = Ball("blue", "large", "up")
15 print (yourBall)
```

init) self = <\_\_main\_\_.Ball object at 0x000002452DBA8910>  
<\_\_main\_\_.Ball object at 0x000002452DBA8910>

init) self = <\_\_main\_\_.Ball object at 0x000002452DC36FD0>  
<\_\_main\_\_.Ball object at 0x000002452DC36FD0>

# Special method : 객체 출력 method

- 특수 method '\_\_str\_\_()'
  - Default로 보여주는 내용 이외에 개발자가 객체에 대해 정의한 내용으로도 출력 가능

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

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("n-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("n-1) 추가된 양념 =", stuff)
47 self.condiments.append(stuff)
48 print("a-2) 최종 양념 =", self.condiments)
49

```

1) main start.myDoc 객체생성

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.myDoc 객체생성")
52 myDog = HotDog()
53 print("Wn2) myDog 객체 정보 출력")
54 print(myDog)
55
56 print("Wn3) Cooking hot dog for 4 minutes.")
57 myDog.cook(4)
58
59 print("Wn4) myDog 객체 정보 출력")
60 print(myDog)
61
62 print("Wn5) Cooking hot dog for 3 more minutes.")
63 myDog.cook(3)
64
65 print("Wn6) myDog 객체 정보 출력")
66 print(myDog)
67
68 print("Wn7) What happens if I cook it for 10 more minutes?")
69 myDog.cook(10)
70 print("Wn8) myDog 객체 정보 출력")
71 print(myDog)
72
73 print("Wn9) Now, I'm going to add some stuff on my hot dog")
74 myDog.addCondiment("ketchup")
75 myDog.addCondiment("mustard")
76
77 print("Wn10) myDog 객체 정보 출력")
78 print(myDog)
79

```

1) main start.myDoc 객체생성  
 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 operator(+, -, \*, /)에 관련된 **특수 메소드(special method)**가 있음

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

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

# Special 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                           |
| <code>divmod(x, y)</code> | <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 operator overload

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)
```

```
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")
```

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.

# Homework 1

- 다음의 vector operator overload code를 class로 구현하시오.

\_\_mul\_\_

\_\_ne\_\_

- 다음 수업 시작 전까지 ecampus로 upload
- 파일명

mul\_이름\_학번.py

ne\_이름\_학번.py

\* 파일이 여러 개일 경우 zip으로 묶어서 제출

# Homework 2

- Class coding 숙제
- “5. Functions” 숙제로 제출하였던 아래의 program들을 class를 이용하여 재작성하여 다음 수업 시작 전까지 ecampus로 upload
- 파일명

fibo\_class\_이름\_학번.py

tree\_class\_이름\_학번.py

hanoi\_class\_이름\_학번.py

day\_class\_이름\_학번.py

\* 파일이 여러 개일 경우 zip으로 묶어서 제출

# Q & A

