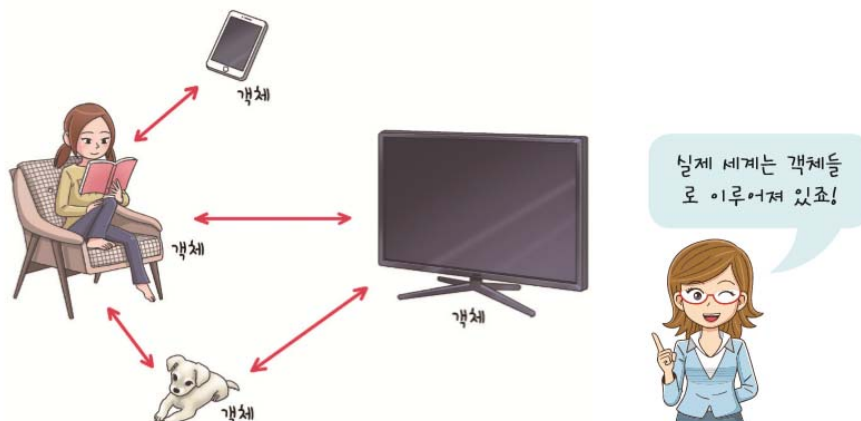


8장. 클래스와 객체

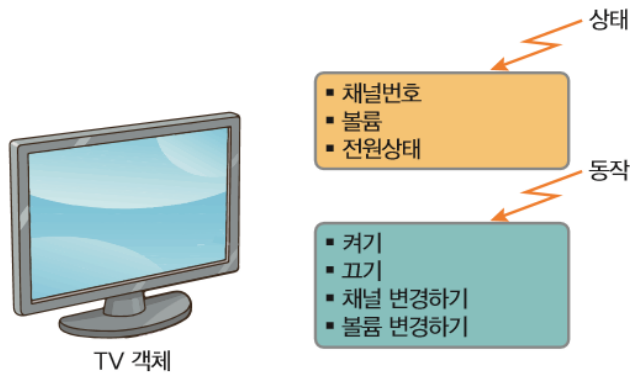
객체지향 프로그래밍

- 객체 지향 프로그래밍(OOP: object-oriented programming)은 우리가 살고 있는 실제 세계가 객체(object)들로 구성되어 있는 것과 비슷하게, 소프트웨어도 객체로 구성하는 방법이다.



객체

- 객체는 상태와 동작을 가지고 있다.
- 객체의 상태(state)는 객체의 속성이다.
- 객체의 동작(behavior)은 객체가 취할 수 있는 동작(기능)이다.

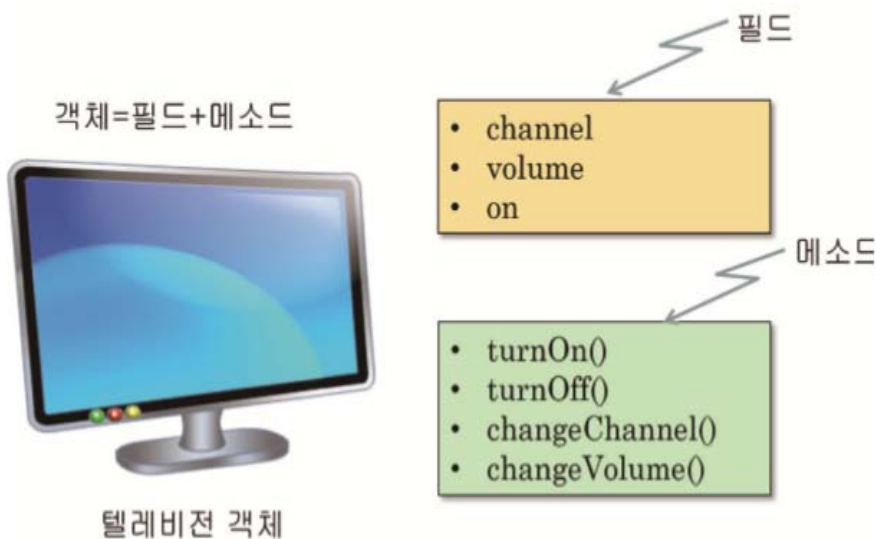


객체는 상태와 동작을 가지고 있습니다.



인스턴스 변수와 메소드

객체=필드+메소드



상태는 인스턴스 변수로, 동작은 메소드로 구현됩니다.



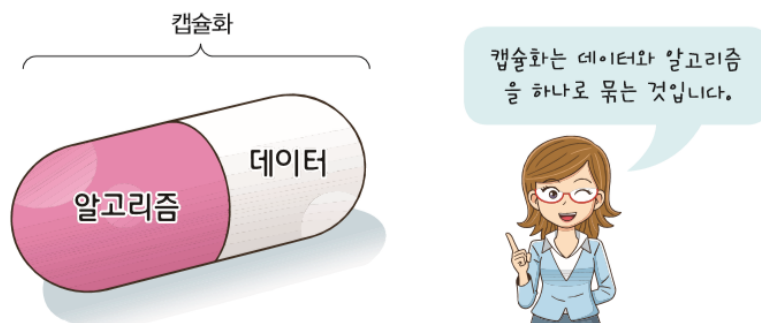
클래스란?

- 객체에 대한 설계도를 클래스(class)라고 한다.
- 클래스로부터 만들어지는 각각의 객체를 그 클래스의 인스턴스(instance)라고 한다.



캡슐화

- 데이터와 알고리즘을 하나로 묶고 공용 인터페이스만 제공하고 구현 세부 사항을 감추는 것은 캡슐화(encapsulation)라고 한다.



클래스 작성하기

전체적인 구조



```
class 클래스 이름 :
```

```
def 메소드1 (self, ...):
```

```
...
```

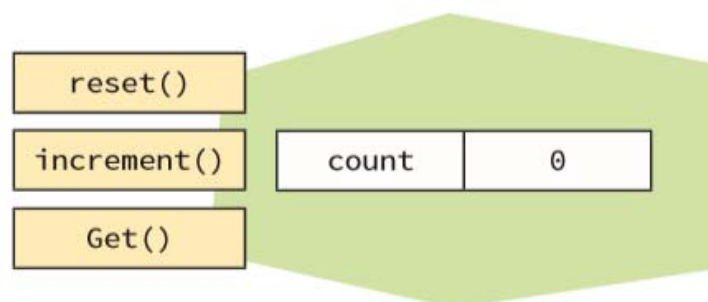
```
def 메소드2 (self, ...):
```

```
...
```

메소드를 정의한다.

클래스의 예

- Counter 클래스를 작성하여 보자. Counter 클래스는 기계식 계수기를 나타내며 경기장이나 콘서트에 입장하는 관객 수를 세기 위하여 사용할 수 있다.



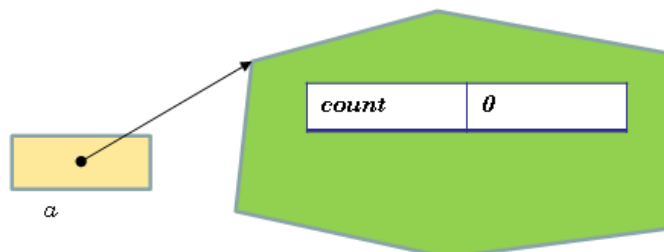
Counter 클래스

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

객체 생성

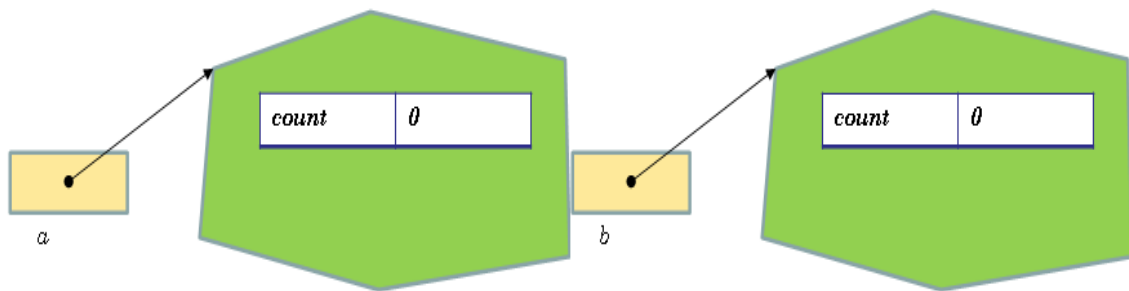
```
a = Counter()  
a.reset()  
a.increment()  
print("카운터 a의 값은", a.get())
```

카운터 a의 값은 1



객체 2개 생성하기

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



생성자

- 생성자(constructor)는 객체가 생성될 때 객체를 기본값으로 초기화하는 특수한 메소드이다.



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
```

메소드 정의

- 메소드는 클래스 안에 정의된 함수이므로 함수를 정의하는 것과 아주 유사하다. 하지만 첫 번째 매개변수는 항상 `self`이어야 한다.

```
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
```


메소드 호출

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

```
t.show()
```

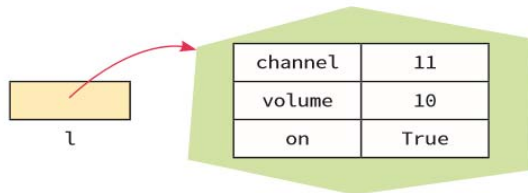
```
t.setChannel(11)
```

```
t.show()
```

```
9 10 True
```

```
11 10 True
```

당분간 소스 코드 다음에는
소스 코드를 설명하는 그림이
등장할 것입니다!



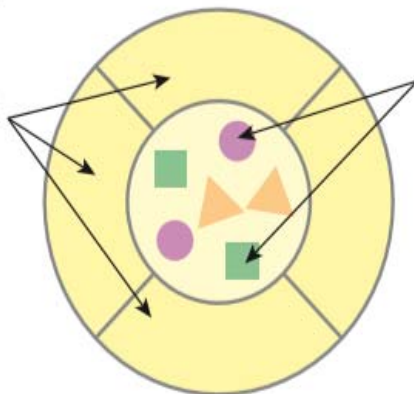
정보 은닉

- 구현의 세부 사항을 클래스 안에 감추는 것

클래스

메소드

변수 선언



변수는 안에 감추고 외부에
서는 메소드들만 사용하도록
하는 것입니다.




```
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'
```

접근자와 설정자

- 인스턴스 변수값을 반환하는 접근자(getters)와 인스턴스 변수값을 설정하는 설정자(setters)가 있다.



```
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()
```

Lab: 원을 클래스로 표현

- 원을 클래스로 구현해보자. 원은 반지름(radius)을 가지고 있다. 원의 넓이와 둘레를 계산하는 메소드도 정의해보자. 설정자와 접근자 메소드도 작성한다.

원의 반지름= 10

원의 넓이= 314.1592653589793

원의 둘레= 62.83185307179586

Solution

```
import math
class Circle:
    def __init__(self, radius=1.0):
        self.__radius = radius

    def setRadius(self, r):
        self.__radius = r

    def getRadius(self):
        return self.__radius

    def calcArea(self):
        area = math.pi*self.__radius*self.__radius
        return area

    def calcCircum(self):
        circumference = 2.0*math.pi*self.__radius
        return circumference

c1=Circle(10)
print("원의 반지름=", c1.getRadius())
print("원의 넓이=", c1.calcArea())
print("원의 둘레=", c1.calcCircum())
```

Lab: 은행 계좌

- 우리는 은행 계좌에 돈을 저금할 수 있고 인출할 수도 있다. 은행 계좌를 클래스로 모델링하여 보자. 은행 계좌는 현재 잔액(balance)만을 인스턴스 변수로 가진다. 생성자와 인출 메소드 `withdraw()`와 저축 메소드 `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("통장에 ", amount, "가 입금되었음")
        return self.__balance

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

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

객체를 함수로 전달할 때

- 우리가 작성한 객체가 전달되면 함수가 객체를 변경할 수 있다.

```
# 사각형을 클래스로 정의한다.
class Rectangle:
    def __init__(self, side=0):
        self.side = side

    def getArea(self):
        return self.side*self.side

# 사각형 객체와 반복횟수를 받아서 변을 증가시키면서 면적을 출력한다.
def printAreas(r, n):
    while n >= 1:
        print(r.side, "\t", r.getArea())
        r.side = r.side + 1
        n = n - 1
```

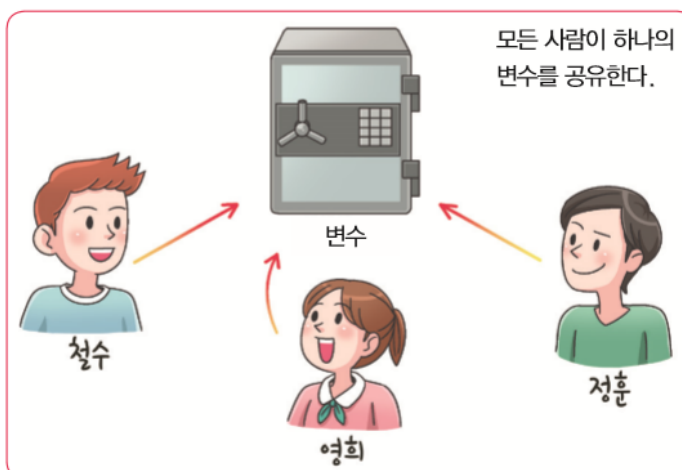
객체를 함수로 전달할 때

```
# printAreas()을 호출하여서 객체의 내용이 변경되는지를 확인한다.  
myRect = Rectangle();  
count = 5  
printAreas(myRect, count)  
print("사각형의 변=", myRect.side)  
print("반복횟수=", count)
```

```
0      0  
1      1  
2      4  
3      9  
4     16  
사각형의 변= 5  
반복횟수= 5
```

정적 변수

- 이들 변수는 모든 객체를 통틀어서 하나만 생성되고 모든 객체가 이것을 공유하게 된다. 이러한 변수를 정적 멤버 또는 클래스 멤버(class member)라고 한다.

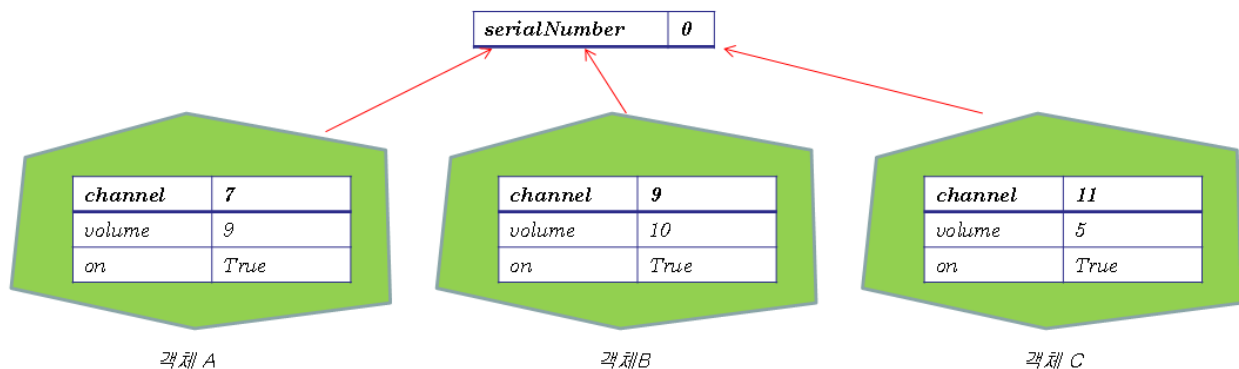


클래스 변수는 클래스당 하나만 생성되어서 모든 객체가 공유합니다.



정적 변수

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



특수 메소드

- 파이썬에는 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 메소드

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

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)
```

파이썬에서의 변수의 종류

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

핵심 정리

- 클래스는 속성과 동작으로 이루어진다. 속성은 인스턴스 변수로 표현되고 동작은 메소드로 표현된다.
- 객체를 생성하려면 생성자 메소드를 호출한다. 생성자 메소드는 `__init__()` 이름의 메소드이다.
- 인스턴스 변수를 정의하려면 생성자 메소드 안에서 **self**. 변수이름 과 같이 생성한다.