

13. 클래스

13.1 파이썬 클래스란

클래스는 새로운 이름 공간을 지원하는 단위이다. 이 이름 공간에는 함수와 변수가 포함될 수 있다. 이러한 점에서는 모듈과 유사하다. 차이점이 있다면, 모듈은 파일 단위로 이름 공간을 구성하는 반면, 클래스는 클래스 이름 공간과 클래스가 생성하는 인스턴스 이름 공간을 각각 갖는다. 클래스 이름 공간과 인스턴스 이름 공간은 유기적인 관계로 연결되어 있으며 상속 관계에 있는 클래스 간의 이름 공간도 유기적으로 연결되어 있다. 클래스를 정의하는 것은 새로운 자료형을 하나 만드는 것이고, 인스턴스는 이 자료형의 객체를 생성하는 것이다.

- 클래스와 이름 공간

우선 클래스 이름 공간의 예부터 살펴보자. 클래스는 하나의 이름 공간이다.

```
>>> class S1:
...     a = 1
...
>>> S1.a
1
>>> S1.b = 2          # 클래스 이름공간에 새로운 이름을 만든다.
>>> S1.b
2
>>> dir(S1)           # 속성 살펴보기
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
 '__getattr__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__',
 '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
 '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'a', 'b']
>>> del S1.b          # 이름 공간 S1에서 이름 b를 삭제한다.
```

클래스 인스턴스는 클래스의 실제 객체이다. 인스턴스 객체도 독자적인 이름 공간을 갖는다. 클래스는 하나 이상의 인스턴스 객체를 생성하는 자료형과 같다.

```
>>> x = S1()          # S1 클래스의 인스턴스 객체 x를 생성한다.
>>> x.a = 10          # 클래스 인스턴스 x의 이름 공간에 이름 a를 만든다.
>>> x.a
10
>>> S1.a              # 클래스의 이름 공간과 인스턴스의 이름 공간은 다르다.
1
>>> y = S1()          # 또 다른 클래스 인스턴스를 생성한다.
>>>
>>> y.a = 300         # 인스턴스 객체 y의 이름 공간에 이름을 만든다.
>>> y.a
300
```

```
>>> x.a                # x이름공간의 이름 a를 확인한다.
10
>>> S1.a               # 클래스 이름공간의 이름 a를 확인한다.
1
```

- 상속

클래스는 상속이 가능하다. 상속받은 클래스 (Subclass, 하위클래스)는 상속해준 클래스(Superclass, 상위클래스)의 모든 속성을 자동으로 물려받는다. 따라서 상속받는 클래스는 물려받는 속성 이외에 추가로 필요한 개별적인 속성만을 정의하면 된다.

```
>>> class A:
...     def f(self):
...         print('base')
...
>>> class B(A):        #클래스 A의 속성을 모두 상속받는다.
...     pass
...
>>> b = B()
>>> b.f()              #클래스 A의 메서드 f가 호출된다.
base
```

- 연산자 중복

클래스는 연산자 중복을 지원한다. 파이썬에서 사용하는 모든 연산자(각종 산술, 논리 연산자, 슬라이싱, 인덱싱 등)의 동작을 직접 정의할수 있다. 연산자를 중복하면 내장 자료형과 비슷한 방식으로 동작하는 클래스를 설계할수 있다.

```
>>> class MyClass:
...     def __add__(self, x):        # 이름 __add__는 + 연산자를 중복한다.
...         print('add {} called'.format(x))
...         return x
...
>>> a = MyClass()
>>> a + 3                          # 더하기 연산자와 중복
add 3 called
3
```

다음은 파이썬에서 클래스에 관련된 용어를 정리한 것이다. '

- 클래스(Class) class 문으로 정의하며, 멤버와 메서드를 가지는 객체이다.
- 클래스 객체(Class Object) 클래스와 같은 의미로 사용한다. 클래스를 종종 특정한 대상을 가리키지 않고 일반적으로 언급하기 위해서 사용하는 반면에, 클래스 객체는 어떤 클래스를 구체적으로 지정하기 위해서 사

용하기도 한다.

- **클래스 인스턴스(Class Instance)** 클래스를 호출하여 생성된 객체이다.
- **클래스 인스턴스 객체(Class Instance Object)** 클래스 인스턴스와 같은 의미이다. 인스턴스 객체라고 부르기도 한다.
- **멤버(Member)** 클래스 혹은 클래스 인스턴스 공간에 정의된 변수이다.
- **메서드(Method)** 클래스 공간에 정의된 함수이다.
- **속성(Attribute)** 멤버와 메서드 전체를 가리킨다. 즉, 이름 공간의 이름 전체를 의미한다.
- **상위클래스(Superclass)** 기반 클래스 라고 하기도 한다. 어떤 클래스의 상위에 있으며 각종 속성을 하위 클래스로 상속해 준다.
- **하위클래스(Subclass)** 파생클래스 라고 한기도 한다. 상위 클래스로부터 상속받는 하위의 클래스를 말한다. 상위 클래스로부터 각종 속성을 상속받으므로 코드와 변수를 공유한다.
- **상속(Inheritance)** 상위 클래스의 속성과 행동을 그대로 받아들이고 추가로 필요한 기능을 클래스에 덧붙이는 것이다. 소프트웨어의 재사용 관점에서 상속은 대단히 중요한 역할을 하며, 프로그램의 개발 시간을 단축해준다. 다른 프로그래밍 기법과 객체지향 프로그래밍을 구분하는 중요한 특징이다. A 클래스를 상위 클래스로 하는 클래스 B를 만들면 B 'is-a' A 관계가 성립한다.
- **다중상속(Multiple Inheritance)** 두 개 이상의 상위 클래스로부터 상속받는 것을 말한다.
- **다형성(Polymorphism)** 상속관계 내의 다른 클래스의 인스턴스들이 같은 메서드 호출에 대해 각각 다르게 반응하도록 하는 기능이다.
- **정보 은닉(Encapsulation)** 메서드와 멤버를 클래스 내에 포함하고 외부에서 접근할수 있도록 인터페이스만을 공개한다. 그리고 다른 속성은 내부에 숨기는 것이다.

13.2 클래스 정의와 인스턴스 객체의 생성

간단한 클래스를 정의해 보자.

```
>>> class Simple:           # 헤더(Header)
...     pass                # 몸체(Body)
...
>>> Simple
<class '__main__.Simple'>   # __main__ 모듈 안의 Simple 클래스
```

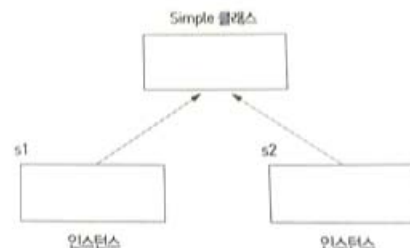
첫줄에 `class` 키워드와 클래스 이름이 나온다. 이줄을 헤더라고 한다. 함수의 첫줄도 헤더이듯이 말이다. 이 문은 선언문이 아닌 실행문으로, 프로그램 중간에 나와도 관계없다. 클래스의 이름은 `Simple`이다. 콜론 `:`을 붙이는 것을 잊지 말아야 한다.

나머지 몸체는 다음 행부터 들여쓰기가 된 상태로 기술한다. `pass` 키워드는 아무 일도 사지 않는, 자리를 채우는 문이다. 이 클래스는 다른 클래스로부터 상속도 받지 않은 것처럼 보이지만 사실 모든 파이썬 클래스는 `object` 클래스를 기반클래스로 한다.

```
>>> Simple.__bases__
(<class 'object'>,)
```

그러면 클래스의 인스턴스 객체를 생성해 보자.

```
>>> s1 = Simple()
>>> s2 = Simple()
>>> s1
<__main__.Simple object at 0x00000215FA7C9C50>
```



클래스의 인스턴스 객체 `s1`과 `s2`를 생성했다. 인스턴스 객체를 생성하는 방법은 마치 함수를 호출하듯이 클래스를 호출하면 된다. 클래스를 호출하면 인스턴스 객체를 생성한 후 해당하는 참조를 반환한다.

앞서 설명했지만 각각의 인스턴스 객체도 독립적인 이름 공간을 갖는다.

```
>>> s1.stack = []           # 인스턴스 객체의 이름 공간 안에 stack을 생성한다.
>>> s1.stack.append(1)      # 값을 추가한다.                  stack을 생성한다.
>>> s1.stack.append(2)
>>> s1.stack.append(3)
>>> s1.stack                # s1.stack의 값을 출력한다.
[1, 2, 3]
>>> s1.stack.pop()          # 값을 읽어 내고 삭제한다.제한다.
3
>>> s1.stack.pop()
2
>>> s1.stack                #최종 s1.stack의 값을 출력한다.
```

[1]

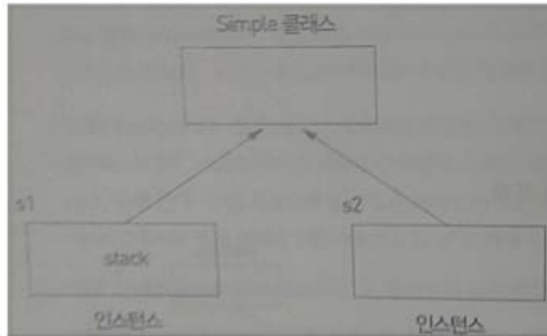
```
>>> s2.stack # s2에는 stack을 정의한 적이 없다.
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
AttributeError: 'Simple' object has no attribute 'stack'
```

```
>>> del s1.stack # s1에서 stack을 삭제한다.
```



동적으로 외부에서 멤버를 생성할 수 있다. 사실 앞서 설명한 것처럼 클래스는 하나의 이름 공간에 불과하날. 클래스의 인스턴스 객체 s1은 클래스 Simple안에 내포된 독립적인 이름 공간을 가지며, 이 이름 공간에서는 동적으로 이름을 설정하는 것이 가능하다.

13.3 메서드의 정의와 호출

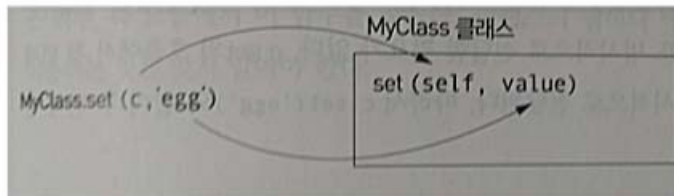
- 일반 메서드의 정의와 호출

앞에서는 외부에서 멤버를 생성하고 사용하는 방법을 설명했지만, 여기서는 클래스 내부에서 멤버를 생성하고 사용하는 방법을 살펴보자. 메서드를 정의하는 방법은 일반 함수를 정의하는 것과 동일하다. 다른점이 있다면 메서드의 첫 번째 인수는 반드시 해당 클래스의 인스턴스 객체이어야 한다. 관례로 self란 이름으로 첫 번째 인수를 선언한다.

```
>>> class MyClass:
...     def set(self, v): # ----- 메서드의 첫 인수 self는 반드시 인스턴스 객체이어야 한다.
...         self.value = v
...     def get(self):
...         return self.value
...
```

메서드를 호출하는 방법에는 두가지가 있는데 첫 번째 방법은 클래스를 이용하여 호출하는 것이다. 이 호출을 언바운드 메서드 호출이라고 부른다.

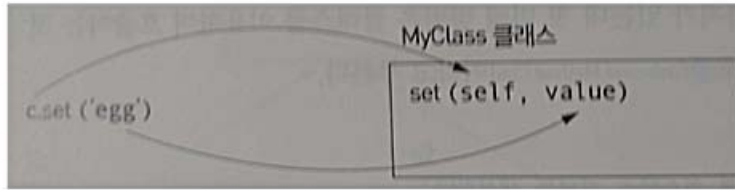
```
>>> c = MyClass() # 인스턴스 객체를 생성한다.
>>> MyClass.set # 메서드를 확인한다.
<function MyClass.set at 0x000002477018A8C8>
>>> MyClass.set(c, 'egg') # 언바운드 메서드 호출이다.
>>> MyClass.get(c)
'egg'
```



(그림) 언바운드 메서드 호출

두 번째 호출방법이 좀더 일반적인데 인스턴스 객체를 이용하여 호출하는 것이다. 이렇게 하면 첫인수로 인스턴스 객체가 자동으로 전달된다. 이 호출을 메서드 호출이라고 부른다.

```
>>> c = MyClass()
>>> c
<__main__.MyClass object at 0x00000247701899B0>
>>> c.set # set() 메서드는 c에 바운드(연결)되어 있다.
<bound method MyClass.set of <__main__.MyClass object at 0x00000247701899B0>>
>>> c.set('egg') # 바운드 메서드 호출이다.
>>> c.get()
'egg'
```



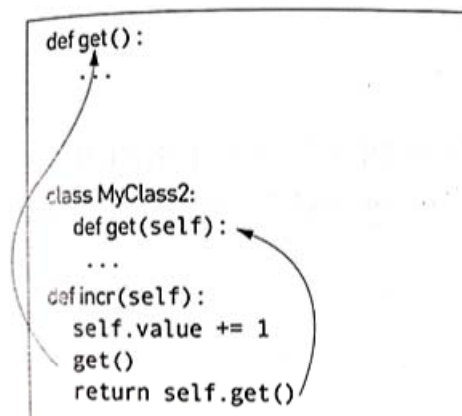
(그림) 바운드 메서드 호출

c.set에 의한 출력내용르 읽어보면 MyClass.set 메서드는 이미 인스턴스 객체 c와 연결되어 있다는 것을 알수 있다. 즉, c.set의 set()메서드는 이미 인스턴스 객체로 c와 연결되어 있다는 의미이다. 따라서 첫 번째 인수를 또 명시적으로 전달할 필요가 없다. c.set의 호출에서 첫 번째 인수 self로 인스턴스 객체 c가 암시적으로 전달된다. 따라서 c.set('egg')와 같이 두 번째 인수부터 전달하면 된다.

- 클래스 내부에서의 메서드 호출

클래스 내부에서 인스턴스 멤버나 클래스 메서드를 호출할때는 인수 self를 이용해야 한다.

```
>>> class MyClass2:
...     def set(self, v):
...         self.value = v
...     def get(self):
...         return self.vlaue
...     def incr(self):
...         self.value += 1      # 인스턴스 멤버를 참조한다.
...         return self.get()   # 메서드를 호출한다.
```



(그림) self를 사용한 경우와 사용하지 않은 경우의 차이

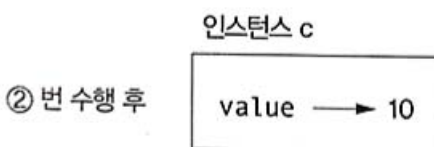
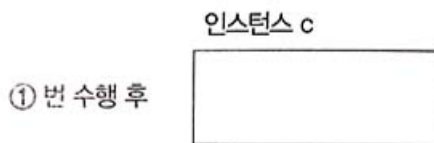
만일 self.get()을 사용하지 않고 그냥 get() 메서드를 호출하면 클래스 내에서도 아니라, 클래스 외부, 특 모듈에서 이 함수를 찾는다. 클래스의 멤버나 메서드를 참조하려면 언제나 self를 이용하는 것을 잊지 말아야 한다.

- 생성자와 소멸자

파이썬은 이름 공간을 관리하는 측면에서 동적인 특성이 강하다. 따라서 클래스의 인스턴스 객체를 생성할 때 인스턴스 멤버가 자동으로 생성되지는 않는다. 앞에서 정의한 클래스 MyClas2를 사용하는 예를 보자.

```
>>> c = MyClass()           # ①
>>> c.set(10)               # ②
>>> c.get()                 # ③
10
```

①번이 실행된 후에는 빈 이름 공간이 만들어진다. 시스템에서 설정하는 여러 이름이 정의되기는 하지만 사용자 정의 이름은 아직 없다. ②번이 실행된 후에야 이름 공간에 value가 만들어진다.



만일 ②번과 ③번의 실행순서가 바뀐다면 어떻게 될까? 당연히 AttributeError 에러가 발생한다. 모든 멤버 변수는 사용하기 전에 먼저 정의되어야 한다. 대부분의 경우, 클래스에서 사용될 멤버들은 인스턴스 객체를 생성하면서 먼저 정의되고 초기화되는 것이 일반적이다.

일반적으로 클래스는 생성자와 소멸자라 불리는 메서드를 정의할수 있다. 생성자는 인스턴스 객체가 생성될 때 초기화를 위해서 자동으로 호출되는 초기화 메서드이고, 소멸자는 인스턴스 객체를 사용하고서 메모리에서 제거할 때 자동으로 호출되는 메서드이다.

파이썬 클래스에서는 생성자와 소멸자를 위해 특별한 이름을 준비해 놓고 있다.

- 생성자 함수의 이름 `__init__`
- 소멸자 함수의 이름 `__del__`

다음 코드에서 `__init__()` 메서드는 인스턴스 객체가 생성되고 나서 자동으로 호출된다. 따라서 인스턴스 멤버 value의 값이 ()으로 초기화된다.

```
>>> class MyClass3:
...     def __init__(self):
...         self.value = 0
...     def set(self, v):
...         self.value = v
```



```

...     def get(self):
...         return self.value
...
>>> c = MyClass3()
>>> c.get()          # 멤버 value의 값이 이미 0으로 초기화되어 있다.
0

```

다음은 `c = MyClass3()` 문을 통해서 인스턴스 객체가 생성되는 과정을 표현한 그림이다. 인스턴스 객체가 생성되고 `__init__()` 메서드로 초기화된 후에 인스턴스 객체의 참조가 변수 `c`로 전달된다.

① 인스턴스 *가 만들어진다.

```
* [ ]
```

② `__init__`가 호출된다.

```
* [self.value = 0]
```

③ 인스턴스가 변수 `c`에 치환된다.

```
c = *
```

다음예를 가지고 생성자와 소멸자가 호출되는 과정을 확인해 보자

```

>>> life = Life()          # 인스턴스 객체가 생성된후 생성자를 호출한다.
Birthday Fri Oct 25 10:47:05 2019
>>> del life              # 소멸자를 호출한 후 인스턴스 객체가 제거된다.
deathday Fri Oct 25 10:47:17 2019

```

Life클래스의 인스턴스 객체가 생성되면 자동으로 `__init__()` 메서드가 호출되어 Birthday를 출력한다. `del` 문에 의해서 참조 횟수의 값이 감소하고 0이되면 `__del__()` 메서드가 호출되고 인스턴스 객체가 제거된다. 소멸자(`__del__`)는 자주 정의되지는 않는다. 대부분의 메모리나 자원관리가 자동으로 이루어지기 때문에 특별한 조치를 취하지 않아도 인스턴스 객체가 제거되면서 자원이 원상 복귀되기 때문이다.

다음은 생성자에 인수가 선언되어 있는 예이다.

```

>>> class Member:
...     def __init__(self, name, nick, birthday):
...         self.name = name
...         self.nick = nick
...         self.birthday = birthday
...     def getName(self):
...         return self.name
...     def getNick(self):
...         return self.nick
...     def getBirthday(self):

```

```

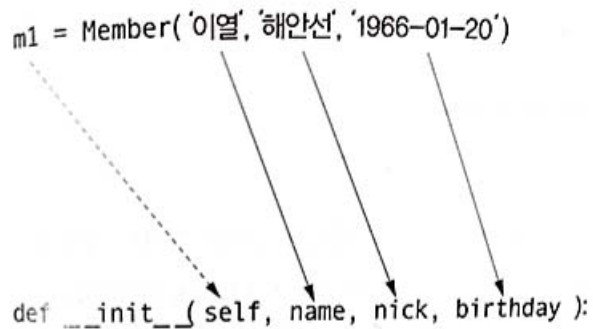
...         return self.birthday
...
>>> m1 = Member('이열', '해안선', '1966-01-20')
>>> m1.getName()
'이열'
>>> m2 = Member(name = '이수민', nick = '홍길동', birthday = '1966-01-20')

```

```

m1 = Member('이열', '해안선', '1966-01-20')

```



The diagram illustrates the argument passing mechanism in Python. It shows the line `m1 = Member('이열', '해안선', '1966-01-20')` and the function definition `def __init__(self, name, nick, birthday):`. Arrows indicate the flow of arguments: a dashed arrow points from the variable `m1` to the `self` parameter, and solid arrows point from the string literals `'이열'`, `'해안선'`, and `'1966-01-20'` to the `name`, `nick`, and `birthday` parameters respectively.

```

def __init__(self, name, nick, birthday):

```

인스턴스 객체를 생성할 때 열거된 인수들은 생성자 인수에 전달된다. `self`는 결과적으로 `m1`과 같은 참조가 되며, 나머지는 순서대로 전달된다. 함수에서와 같은 사용 방법이 적용된다.

13.4 클래스 멤버와 인스턴스 멤버

- 클래스 멤버와 인스턴스 멤버 구분하기
- 멤버에는 클래스 멤버와 인스턴스 멤버 두가지가 있다.
- 클래스 멤버는 클래스의 이름 공간에 생성된다.
- 인스턴스 멤버는 인스턴스 객체의 이름 공간에 생성된다.
- 클래스 멤버는 모든 인스턴스 객체에 의해서 공유될 수 있다.
- 인스턴스 멤버는 각각의 인스턴스 객체 내에서만 참조된다.

두 가지의 멤버를 정의하는 예를 보자.

```
>>> class Var:
...     c_mem = 100          # 클래스 멤버
...     def f(self):
...         self.i_mem = 200    # 인스턴스 멤버
...     def g(self):
...         return self.i_mem.self.c_mem
```

클래스 멤버는 메서드 바깥에 정의한다. 인스턴스 멤버는 메서드 내부에서 `self`를 이용하여 정의한다. 클래스 내부에서 멤버들을 참조할때는 `self.c_mem`, `self.i_mem`과 같이 한다. 외부에서 참조할때는 다음과 같은 형식으로 호출할수 있다.

- 클래스 멤버 클래스, 멤버 혹은 인스턴스.멤버
- 인스턴스 멤버 인스턴스.멤버

인스턴스.멤버 형식을 사용할 때 인스턴스 객체의 이름 공간에 멤버가 없으면 클래스의 멤버로 인식한다. 따라서 다음과 같이 `Var.c_mem`를 `v1.c_mem`로 참조하는 것이 가능하다.

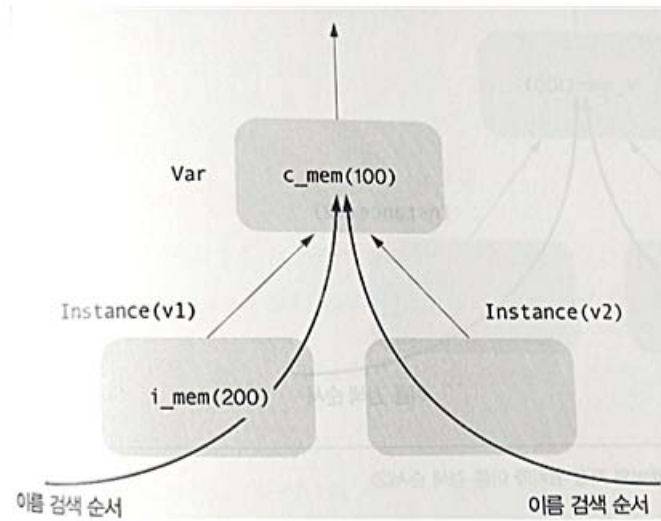
```
>>> Var.c_mem          # 클래스 객체를 통하여
100
>>> v1 = Var()
>>> v1.c_mem           # 인스턴스 객체를 통하여
100
>>> v1.f()             # 인스턴스 멤버 I_mem를 생성
>>> v2 = Var()
```

정리하면 `self.c_mem` 형식이나 인스턴스.멤버 형식으로 멤버를 참조할 때, 검색 순서는 다음과 같다.

- ① 먼저 인스턴스 객체의 이름 공간에서 멤버를 참조한다.
- ② 만일 인스턴스 객체의 이름 공간에 멤버가 없으면 클래스의 멤버를 참조한다.

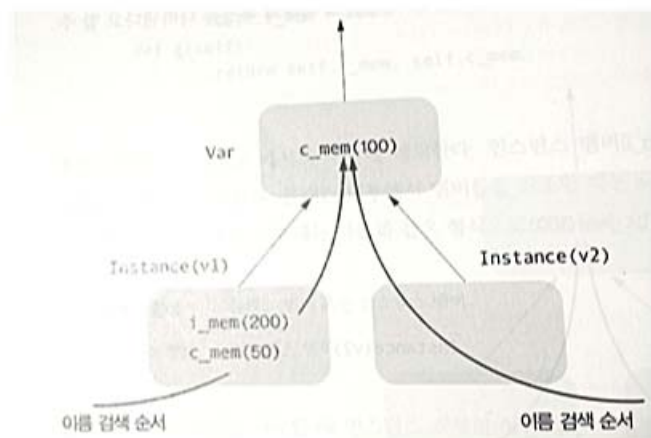
클래스 멤버는 클래스의 모든 인스턴스 객체가 공유하는 멤버이고, 인스턴스 멤버는 각각의 인스턴스 객체가

별로 가지고 있는 멤버이다. 즉, 각 인스턴스 객체의 특성을 나타낸다고 할수 있다.



```
>>> v1.c_mem          # 클래스 멤버를 참조한다
100
>>> v2.c_mem          # 클래스 멤버를 참조한다.
100
>>> v1.c_mem = 50     # 인스턴스 객체의 이름 공간에 c_mem을 생성한다.
>>> v1.c_mem          # 인스턴스 멤버를 참조한다. 앞 코드와 비교하기 바란다.
50
>>> v2.c_mem          # 인스턴스 멤버가 없으므로 클래스 멤버를 참조한다.
100
>>> Var.c_mem         # 클래스 멤버를 참조한다.
100
```

처음의 v1.c_mem은 클래스 멤버를 참조하지만 나중의 v1.c_mem은 인스턴스 멤버가 된다.



- 사용 가능한 멤버 고정하기 : `__slots__` 속성

클래스에 정의한 이름 이외의 속성을 맞출 수 없게 하는 것이 필요하다면 `__slots__` 속성을 이용한다. 다음 예는 클래스 `Person`에 `name`과 `tel` 이외의 속성은 지정할 수 없도록 한다. 클래스 멤버 `__slots__`는 리스트로 설정 가능한 속성의 이름을 갖는다. `__slots__` 속성에 등록되지 않은 이름은 사용할 수 없게 된다. 이 속성이 사용되는 경우에는 이름 공간을 표현하는 `__dict__`는 사용되지 않는다.

```
>>> class Person:
...     __slots__ = ['name','tel']
...
>>> m1 = Person()
>>> m1.name = '이강성'           # __slots__ 속성에 등록된 속성
>>> m1.tel = '5284'             # __slots__ 속성에 등록된 속성
>>> m1.address = '서울'         # __slots__ 속성에 등록되지 않은 속성
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Person' object has no attribute 'address'
```