Chap26. ~ Chap28.

마영전서버유닛
임형진

Chap26. 믹스인 유틸리티 클래스에만 다중 상속을 사용하자

Chap26.

Mixin 믹스인?

Chap26.

변수는 없고 함수만 모아둔 클래스

결국 상속해서 사용하는데 변수가 없고 함수만 있으니 충돌 가능성이 적음

Python에 정의된 기본 타입은 아님

디자인 패턴 처럼 권장 사용법 정도임

별다른 제약은 없음

사용하는 변수가 없다면 Mixin클래스도 객체화 할 수 있고 사용도 가능함

Interface 처럼 속성을 강제할 수 있음

근데 이거보다 뒤에 나오는 ABC를 사용하는게 더 좋음

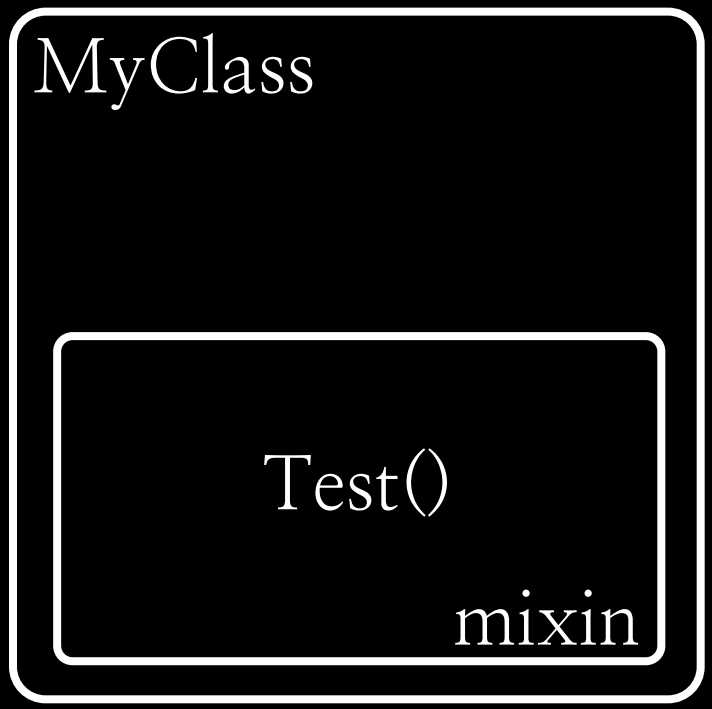
Chap26.

사용법

코드를 보며 이해해 보자

Chap26.

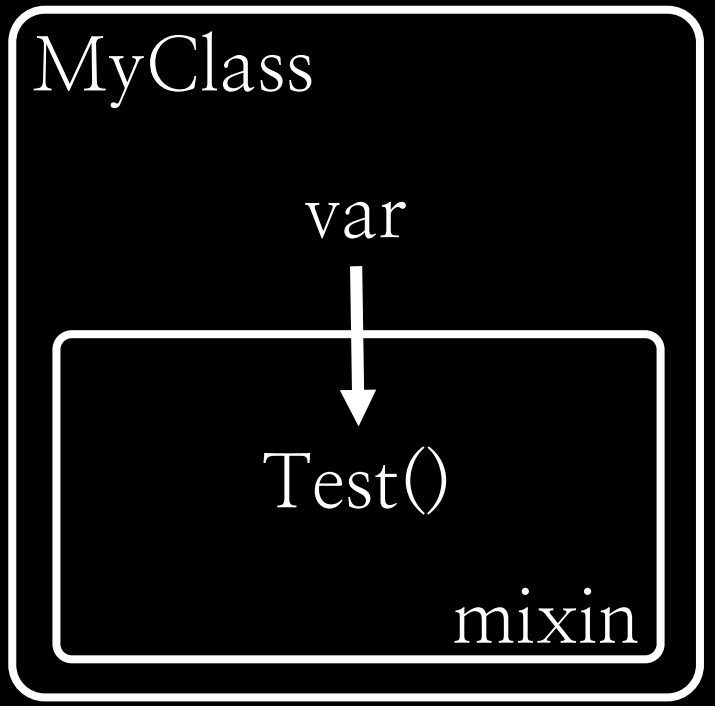
```
1  class mixin(object):
2
3    def Test(self):
4      print ("Call Test!")
5
6  class MyClass(mixin):
7        pass
8
9   myCls = MyClass()
10  myCls.Test()
```



Chap26.

```
1  class mixin(object):
2
3      def Test(self):
4          print ("var is %d" % self.var)
5
6
7  class MyClass(mixin):
8      var = 1
9
10 myCls = MyClass()
11 myCls.Test()
```

없으면 오류발생 (하위 클래스에 var 속성 강제)



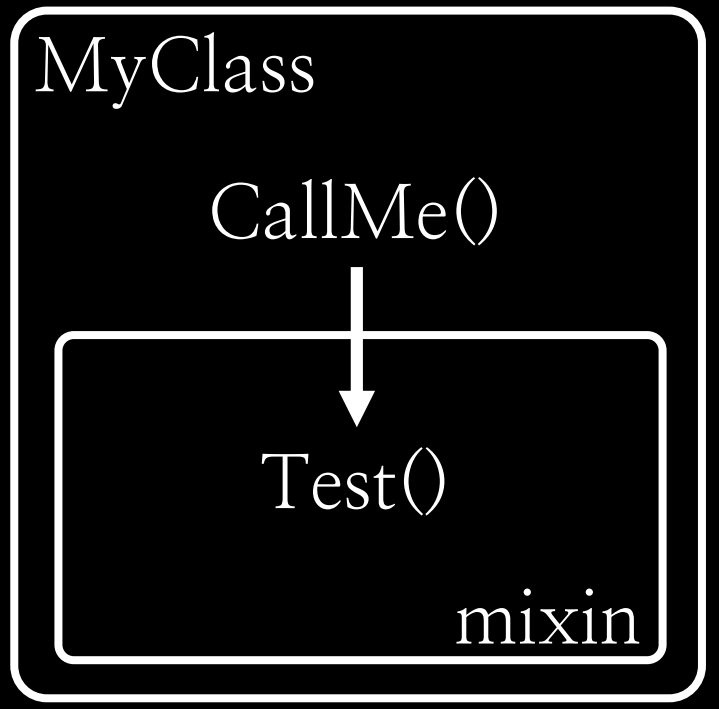
Chap26.

혹시..

이것도 되나?

Chap26.

```
1  class mixin(object):
2
3      def Test(self):
4          self.CallMe()
5
6  class MyClass(mixin):
7                    없으면 오류발생
8      def CallMe(self):
9          print("Call Me!")
10
11
12 myCls = MyClass()
13 myCls.Test()
```



Chap26.

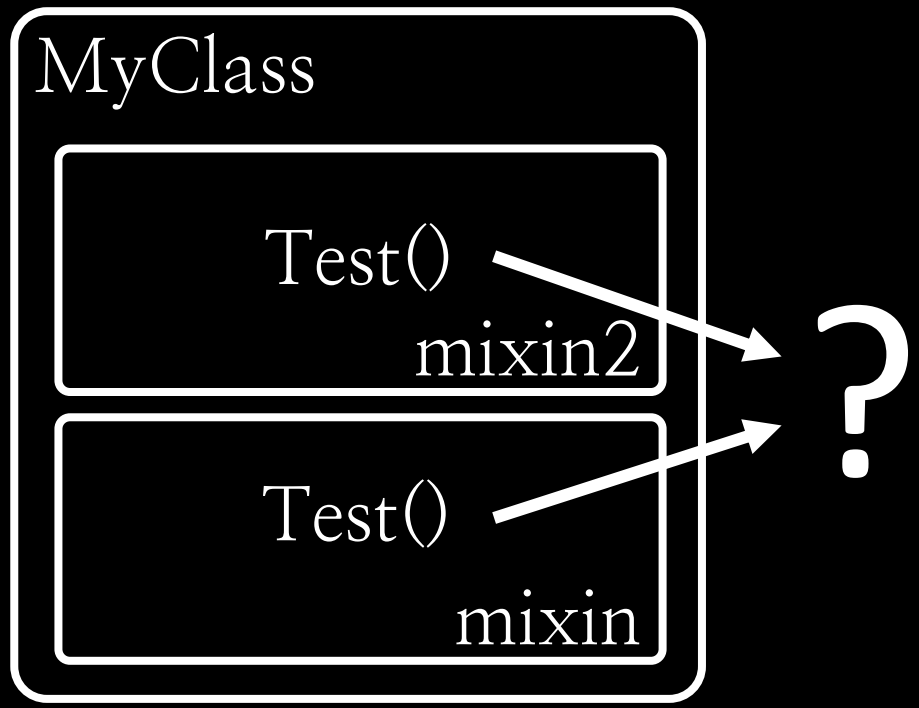
```python
class mixin(object):

    def Test(self):
        print ("Call Test")

class mixin2(object):

    def Test(self):
        print ("Call Test2")

class MyClass(mixin2, mixin):
    pass
```



이러면 다를게 없지 않나…

Chap26.

결국 클래스이기 때문에
여전히 다중 상속의 문제가 발생 할 수 있음

알아서 잘~ 사용해야함

p.118 클래스 메소드 관련 내용은 생략

Chap27. 공개 속성보다는 비공개 속성을 사용하자

Python은 이름을 변경해 Private 변수를 숨긴다

__〈FieldName〉 → _〈ClassName〉__〈FieldName〉

변경된 이름을 알면 접근할 수 있다.

Chap27.

```
 1  class MyClass(object):
 2
 3      def __init__(self):
 4          self.__pField = 1
 5
 6      def Test(self):
 7          print(self.__pField)
 8
 9  pTest = MyClass()
10  pTest.Test()
11  print(pTest.__dict__)
```

```
root@f32117d75056:~/slideCode# python3 test5.py
1
{'_MyClass__pField': 1}
```

pTest._MyClass__pField 로 접근가능

Chap27.

이 책에서만 그런지 모르겠는데

부모 클래스 멤버의 접근을 권장하는 느낌..
OOP 관점으로 이해하면 난해하다.

Chap27.

공개 속성보다는 ~~비공개~~ 보호 속성을 사용하자

_protect_field
“이 속성은 사용할때 조심..”

서브클래스와 이름이 충돌할 수 있는 때만 비공개 속성을 사용하자

Chap28. 커스텀 컨테이너 타입은 collections.abc의 클래스를 상속받게 만들자

Abstract Base Classes

Metaclass를 ABCMeta로 변경하고
함수에 @abstractmethod 를 붙이면 서브 클래스에서는
해당 함수를 오버라이드 하기전에 인스턴스화 할 수 없다.

코드를 보며 이해해 보자

Chap28.

```
1 from collections.abc import Sequence
2
3 class BadType(Sequence):
4     pass
5
6 foo = BadType()
```

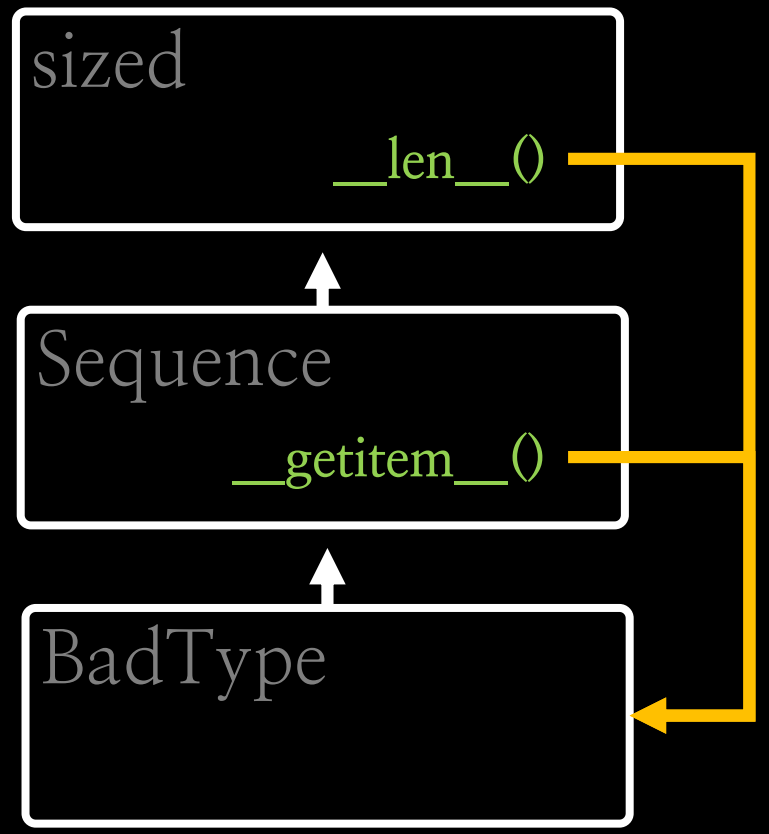
```
TypeError: Can't instantiate abstract class BadType with abstract methods __getitem__, __len__
```

Chap28.

```
1  class Sequence(Sized, Iterable, Container):
2
3      """All the operations on a read-only sequence.
4
5      Concrete subclasses must override __new__ or __init__,
6      __getitem__, and __len__.
7      """
8
9      __slots__ = ()
10
11     @abstractmethod
12     def __getitem__(self, index):
13         raise IndexError
14
15     ...
```

Chap28.

```
2  class Sized(metaclass=ABCMeta):
3
4      __slots__ = ()
5
6      @abstractmethod
7      def __len__(self):
8          return 0
```



Chap28.

collections.abc에 정의된 인터페이스를 사용하면
본래 컨테이너에 있는 기능이 빠짐없이 동작하게 할 수 있다.

최대한 자유롭게 사용 할 수 있게 만들어
사용자가 알아서 잘 이해하고 쓸꺼야.

끝