Programowanie obiektowe

Metody statyczne i klasowe

Paweł Daniluk

Wydział Fizyki

Jesień 2013

W poprzednich odcinkach...

Klasy – kategorie obiektów

Obiekty – instancje klas

Przynależność do klasy określa zakres odpowiedzialności obiektów.

Każdy obiekt należy do pewnej klasy.

Klasa określa funkcjonalności (metody) obiektów.

Każdy obiekt odpowiada za wartości swoich atrybutów.

Metody określone przez klasę odwołują się do atrybutów przechowywanych w obiekcie.

Metody mogą być dziedziczone.

W poprzednich odcinkach...

Klasy – kategorie obiektów

Obiekty – instancje klas

Przynależność do klasy określa zakres odpowiedzialności obiektów.

Każdy obiekt należy do pewnej klasy.

Klasa określa funkcjonalności (metody) obiektów.

Każdy obiekt odpowiada za wartości swoich atrybutów.

Metody określone przez klasę odwołują się do atrybutów przechowywanych w obiekcie.

Metody mogą być dziedziczone.

Czy klasy mogą być obiektami?

Klasy są obiektami (w Pythonie)

Klasy mogą mieć atrybuty

```
>>> class A:
...     pass
...
>>>
>>> A.class_attr="classA"
>>>
>>> a.class_attr
'classA'
```

Atrybuty klas są dziedziczone

```
>>> class B(A): pass
...
>>> B.class_attr
'class A'
>>>
```

Przestrzenie nazw

Odwołanie do składowej

```
obiekt.atrybut
obiekt.metoda()
Klasa.atrybut
```

Kolejność przeszukiwania

1. Obiekt
2. Klasa obiektu
3. Nadklasa
4. Kolejne nadklasy ...

Przeszukiwanie odbywa się do skutku. Atrybuty obiektu przysłaniają atrybuty klas. Atrybuty podklasy przysłaniają atrybuty nadklasy.

Przykład

A

```
attr1="class A"
attr2="class A"
```

B

```
attr2="class B"
```

obj1

```
attr1="obj 1"
```

obj2

```
attr2="obj 2"
```

Uwaga

Przypisanie zawsze tworzy atrybut w najbliższym zasięgu przeszukiwania.

Przykład

```
>>> class A: pass
...
>>> class B(A): pass
...
>>> A.attr="class A"
>>> A.attr
'class A'
>>> B.attr
'class A'
>>> B.attr="class B"
>>> A.attr
'class A'
>>> B.attr
'class B'
```

Słownik __dict__

Atrybuty własne obiektu (klasy) znajdują się w słowniku __dict__.

Przykład

```
>>> B.__dict__
'__module__': '__main__', '__doc__': None, 'attr': 'class B'
>>> class C(B): pass
...
>>> C.__dict__
'__module__': '__main__', '__doc__': None
```

Zastosowania atrybutów klasowych

- Wartości charakteryzujące podklasy
- Wartości wspólne dla wszystkich instancji (stałe i parametry konfiguracyjne)
- Zmienne "globalne"

Wartości charakteryzujące podklasy

Czasami stosując wzorzec *Template method* wystarczy w podklasie użyć atrybutu.

Zastosowania atrybutów klasowych c.d.

Przykład

```
class Zwierze:
    def daj_glos(self):
        print self.glos

class Pies(Zwierze):
    glos="Woof, woof"

class Swinka(Zwierze):
    glos="Oink, oink"

class LewHoward(Zwierze):
    glos="Roark"
    def daj_glos(self):
        Ziemia.trzes_sie()
        Zwierze.daj_glos(self)
```

Zastosowania atrybutów klasowych c.d.

Wartości wspólne dla wszystkich instancji (stałe i parametry konfiguracyjne)

Nigdy nie umieszcza się w kodzie stałych, ani parametrów jako wartości, ponieważ ich zmiana wiązałaby się z koniecznością odnalezienia wszystkich wystąpień. Ze względów projektowych dobrze jest umieszczać stałe i parametry jak najbliżej metod, które z nich korzystają.

Zmienne "globalne"

Przykłady:

- licznik utworzonych instancji
- ostatnio obliczona wartość
- kontener na instancje
- połączenie z bazą danych

Metody w klasach

Pytanie (przewrotne)

Skoro funkcje w Pythonie są wartościami, a klasy mogą mieć atrybuty, to czemu nie tworzyć atrybutów klas, które są funkcjami?

Odpowiedź

Bo nie trzeba. Są metody statyczne.

Metody statyczne

Metoda statyczna różni się od zwykłej brakiem argumentu `self`. W związku z tym może być wywoływana dla klasy.

Definicję metody statycznej poprzedza się dekoratorem `@staticmethod`.

Przykład

```
class A:
    attr="class A"

    @staticmethod
    def m():
        print A.attr
```

W metodach statycznych do atrybutów klasy można się odwoływać przez jej nazwę.

Singleton

Singleton to wzorzec projektowy, który polega na ograniczeniu liczby instancji obiektu danej klasy to jednej i zapewnieniu łatwego dostępu do jedynej instancji. Stosuje się go, jeżeli z różnych powodów wiele instancji wzajemnie by sobie przeszkadzało.

Przykłady

- fabryki obiektów
- stan aplikacji
- wszelkie sytuacje, gdy zmienne globalne są naprawdę konieczne i mają skomplikowaną strukturę

Czasami zamiast prawdziwego singletona wystarczy klasa z atrybutami i metodami statycznymi.

Singleton – przykład

```
class Singleton:
    _instance=None

    @staticmethod
    def get_instance():
        if Singleton._instance==None:
            Singleton._instance=Singleton()

        return Singleton._instance
```

Singleton – przykład

```
class Singleton:
    _instance=None

    @staticmethod
    def get_instance():
        if Singleton._instance==None:
            Singleton._instance=Singleton()

        return Singleton._instance
```

W Pythonie takie rozwiązanie nie gwarantuje, że kolejne instancje nie zostaną stworzone bezpośrednio. Można wprowadzić zabezpieczenie w metodzie `__init__`.

Singleton – przykład

```
class Singleton:
    _instance=None

    @staticmethod
    def get_instance():
        if Singleton._instance==None:
            Singleton._instance=Singleton()

        return Singleton._instance
```

W Pythonie takie rozwiązanie nie gwarantuje, że kolejne instancje nie zostaną stworzone bezpośrednio. Można wprowadzić zabezpieczenie w metodzie `__init__`.

Singletonów należy używać wyłącznie, gdy jest to naprawdę konieczne – da się udowodnić, że nie można inaczej.

Metody statyczne – ostrzeżenia

Metody statyczne w zasadzie nie różnią się niczym od funkcji zdefiniowanych poza klasą, ale pozwalają na lepszą organizację kodu.

Metody statyczne – ostrzeżenia

Metody statyczne w zasadzie nie różnią się niczym od funkcji zdefiniowanych poza klasą, ale pozwalają na lepszą organizację kodu.

Java

W Javie używanie metod statycznych jest koniecznością, bo nie można definiować funkcji. W Pythonie jest sens ich używać wyłącznie, gdy poprawia to czytelność kodu...

Metody statyczne – ostrzeżenia

Metody statyczne w zasadzie nie różnią się niczym od funkcji zdefiniowanych poza klasą, ale pozwalają na lepszą organizację kodu.

Java

W Javie używanie metod statycznych jest koniecznością, bo nie można definiować funkcji. W Pythonie jest sens ich używać wyłącznie, gdy poprawia to czytelność kodu...

... albo chcemy wywoływać metody statyczne również dla instancji obiektu.

Metody statyczne – ostrzeżenia

Metody statyczne w zasadzie nie różnią się niczym od funkcji zdefiniowanych poza klasą, ale pozwalają na lepszą organizację kodu.

Java

W Javie używanie metod statycznych jest koniecznością, bo nie można definiować funkcji. W Pythonie jest sens ich używać wyłącznie, gdy poprawia to czytelność kodu...

... albo chcemy wywoływać metody statyczne również dla instancji obiektu.

Dziedziczenie

Metody statyczne powodują kłopoty przy dziedziczeniu, jeżeli atrybuty, do których się odnoszą są przysłonięte w podklasie.

Metody statyczne – dziedziczenie

```
>>> class A:
...     attr="class A"
...
...     @staticmethod
...     def print_attr():
...         print A.attr
...
>>> A.print_attr()
class A
>>>
>>> class B(A):
...    attr="class B"
...
>>> B.print_attr()
class A
>>>
```

Gdyby była możliwość przekazania klasy jako argumentu metody, można byłoby odwoływać się do jej atrybutów.

Metody klasowe

Metoda klasowa ma argument `cls` działający analogicznie do `self`.

Definicję metody statycznej poprzedza się dekoratorem `@classmethod`.

Przykład

```
class A:
    attr="class A"

    @classmethod
    def m(cls):
        print cls.attr
```

Metody klasowe – dziedziczenie

```
>>> class A:
...     attr="class A"
...
...     @classmethod
...     def print_attr(cls):
...         print cls.attr
...
>>> A.print_attr()
class A
>>>
>>> class B(A):
...    attr="class B"
...
>>> B.print_attr()
class B
>>>
```

Factory methods – jeszcze raz

Metody klasowe znakomicie nadają się do tworzenia *factory methods*.

Przykład

```
class Abstract:
    @classmethod
    def create(cls):
        cls.count+=1
        return cls()

class A(Abstract):
    count=0
    def __init__(self):
        print "init A"

class B(A):
    count=0
```

Factory methods – jeszcze raz c.d.

Test

```
>>> A.create()
init A
<A instance at 0x10a5c0cf8>
>>> A.create()
init A
<A instance at 0x10a5c0d88>
>>> B.create()
init A
<B instance at 0x10a5c0cf8>
>>> A.count
2
>>> B.count
1
```

Zadanie 1 – Numerowanie instancji

Zadanie

Zaimplementuj abstrakcyjną klasę umożliwiającą tworzenie instancji podklas zawierających atrybut `order`, który służy do numerowania kolejnych instancji podklas (każdej niezależnie).

Zadanie 2 – Multiton

Multiton jest klasą, która może mieć ściśle określoną liczbę instancji. Wszystkie instancje powinny być dostępne poprzez podanie klucza. Zaimplementuj hierarchę multitonów, które mogą się różnić się zbiorami kluczy.