Politechnika Wrocławska

Python
wstęp do programowania
dla użytkowników WCSS

Dr inż. Krzysztof Berezowski
Instytut Informatyki, Automatyki i Robotyki
Politechniki Wrocławskiej

Programowanie obiektowe

KLASA I OBIEKT

Klasa i obiekt

- Programowanie obiektowe to mechanizm pozwalający na definiowanie złożonych abstrakcyjnych hierarchii typów danych.
- Intencją jest dostarczenie mechanizmów:
  - oddzielania abstrakcyjnego interfejsu od szczegółów implementacji,
  - powtórne wykorzystane podobieństw,
  - przejrzyste uwypuklenie różnic,
  - skuteczne poddanie typów użytkownika reżimowi systemu typów języka.

Klasa i obiekt

- **Klasy są typami danych,**
- **obiekty instancjami tychże typów**

```
class Foo:
    pass

foo = Foo()
print Foo
print foo
print type(Foo)
print type(foo)
```

Problems | Console

```
<terminated> /Users/kberezow/Documents/Python WCSS/python5/classes/src/p1_class
__main__.Foo
<__main__.Foo instance at 0x1004a9200>
<type 'classobj'>
<type 'instance'>
```

Klasa i obiekt

- **Programowania obiektowego naturalnie używamy tam gdzie można wyróżnić**
  - **abstrakcyjne własności danych,**
  - **abstrakcyjne operacje na danych, oraz**
  - **hierarchiczne zależności między typami.**

Czyli wszędzie!

Abstrakcja w programowaniu

- **Oddzielenie interfejsu od implementacji co pozwala ukryć jej szczegóły**
- **Umożliwienie kontrolowanego dostępu do implementacji poprzez jednoznacznie zdefiniowany przejrzysty interfejs opisany spójnym kontraktem**
- **Prostota i przejrzystość składniowa**

Abstrakcja w programowaniu

- **Oddzielenie interfejsu od implementacji co pozwala ukryć jej szczegóły**
- **Umożliwienie kontrolowanego dostępu do implementacji poprzez jednoznacznie zdefiniowany przejrzysty interfejs opisany spójnym kontraktem**
- **Prostota i przejrzystość składniowa (dlatego wyróżniamy podejście obiektowe)**

Abstrakcja

```
import math
class Complex():
    def __init__(self, real, imag = 0):
        self._real = real
        self._imag = imag

    def real(self):
        return self._real

    def imag(self):
        return self._imag

    def magnitude(self):
        return math.sqrt(self._real**2 + self._imag**2)

    def phase(self):
        return math.atan(self._imag/self._real)

c1 = Complex(1)
c2 = Complex(1, 2)

print c1.magnitude()
print c2.real(), c2.imag(), c2.magnitude(), c2.phase()
```

Problems | Console

```
<terminated> /Users/kberezow/Documents/Python WCSS/python5/classes/src/p3_class.py
1.0
1 2 2.2360679775 1.10714871779
```

Osadzenie w systemie typów

- **Klasa jest odrębym typem w systemie typów podlegającym reżimowi systemu typów.**
- **Obiekt jest wartością konkretnego typu bez względu na podobieństwo strukturalne**

```
class Foo: pass
class Bar: pass

f = Foo()
b = Bar()

print f == b
print isinstance(f, Foo), isinstance(b, Bar), isinstance(f, Bar), isinstance(b, Foo)
```

Problems | Console

```
<terminated> /Users/kberezow/Documents/Python WCSS/python5/classes/src/a3_class.py
False
True True False False
```

Czytelność intencji

```
class Point2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def get_x(self):
        return self.x

    def get_y(self, ):
        return self.y

    def dist_to(self, p):
        return abs(math.sqrt((self.get_x() - p.get_x())**2 + (self.get_y() - p.get_y())**2))

    def mirror(self):
        return Point2D(-self.x, self.y)

    def __str__(self):
        return "Point2D(%s, %s)" % (self.x, self.y)

center = Point2D(0,0)
p1 = Point2D(1, 1)
p2 = Point2D(2, 2)

print p1.dist_to(p2), p2.dist_to(center)
print p2.mirror()
```

Problems Console

```
<terminated> /Users/kberezow/Documents/Python WCSS/python5/classes/src/c1_point.py
1.41421356237 2.82842712475
Point2D(-2, 2)
```

- zdefiniować settery i gettery, pokazać property
- pokazać sposób konstruowania i konsekwencje abstrakcji

TU PRZYKŁAD

Klasa i obiekt

- Technicznie klasa to:
  - definicja klasy (nazwa, pochodzenie)
  - dane - pola składowe
  - metody - funkcje składowe
  - inicjacja obiektu - konstruktor
  - dekonstrukcja obiektu - destruktor

Klasa jako struktura

- **Ponieważ język jest dynamiczny a klasy otwarte możliwe jest tworzenie pól:**

```
class Osoba:
    pass

e = Osoba()

e.imie = "Jan"
e.nazwisko = "Kowalski"
e.plec = "M"
e.wiek = 25

print e.imie, e.nazwisko
```

- **oznacza to że każdemu obiektowi dowolnej klasy możemy tworzyć nowe pola dynamicznie!**

Zasada: wszystkie składowe są publiczne

- Skoro oddzielamy interfejs od implementacji implementacja nie musi/powinna być publiczna
- Jednak klasa w Pythonie nie jest kapsułkowana (niektórzy uważają, że to źle)

```
class Foo:
    def __init__(self, x):
        self.x = x

    def get_x(self):
        return self.x

f = Foo("Ala ma kota")

print f.x
print f.get_x()
```

Enkapsulacja przez gmatwanie nazw

- **Namiastką enkapsulacji jest konwencja nazw:**

```
class Foo:
    def __init__(self, x):
        self.__x = x

    def get_x(self):
        print self.__x

f = Foo("Ala ma kota")
f.get_x()
f.__x
```

Problems | Console

```
<terminated> /Users/kberezow/Documents/Python WCSS/python5/classes/src/a9_encapsulation.py
Ala ma kota
Traceback (most recent call last):
  File "/Users/kberezow/Documents/Python WCSS/python5/classes/src/a9_encapsulation.py",
    f.__x
AttributeError: Foo instance has no attribute '__x'
```

Dziedziczenie

- **Służy konstrukcji hierarchii typów pokrewnych**

```
class Parent:
    def __init__(self):
        self.name = "Parent"

    def getname(self):
        return self.name

class Child(Parent):
    def __init__(self):
        self.name = "Child"

p = Parent()
c = Child()

print "This is", p.getname()
print "This is", c.getname()
```

Problems | Console

<terminated> /Users/kberezow/Documents/Python WCSS/pyt

```
Parent
Child
```