Politechnika Wrocławska

Programowanie obiektowe

W11FTE-SI0080G (INP001129WL)

rok akademicki 2023/24

semestr letni

Wykład 9

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 220

Plan

- Dziedziczenie
- Wprowadzenie do zagadnienia atrybutów i metod statycznych

Politechnika Wrocławska

Dziedziczenie

- Prosty przykład pokazuje utworzenie dwóch klas pochodnych reprezentujących ptaki oraz ssaki dziedziczących po klasie bazowej reprezentującej zwierzęta

Politechnika Wrocławska

Dziedziczenie

dziedziczenie01.py

```python
# -*- coding: utf-8 -*-
"""
Prosty przykład ilustrujący tworzenie hierarchii klas.
"""

# definicja klasy bazowej reprezentującej zwierzę
class Animal:
    # klasa implementuje konstruktor
    # jedynym atrybutem obiektu jest m (masa)
    def __init__(self, m):
        self.m    = m

    # klasa implemenutuje metodę moves(),
    # która wyświetla ogólną informację o zwierzętach
    def moves(self):
        print("Zwierzęta się przemieszczają.")
```

Politechnika Wrocławska

Dziedziczenie

dziedziczenie01.py

```
18  # definicja klasy pochodnej
19  class Bird(Animal):
20      # klasa pochodna ma wszystkie cechy klasy bazowej,
21      # a dodatkowo implementuje metodę flights(),
22      # która wyświetla informację o ptakach
23      def flights(self):
24          print("Ptaki latają.")
25
26  # definicja klasy pochodnej
27  class Mammal(Animal):
28      # klasa pochodna implementuje metodę runs(),
29      # która wyświetla informację o ssakach
30      def runs(self):
31          print("Ssaki biegają.")
```

Politechnika Wrocławska

Dziedziczenie

dziedziczenie01.py

```
33  # utworzenie obiektu klasy Animal
34  a = Animal(1.)
35  print(a.m)
36  a.moves()
37
38  # klasa Bird dziedziczy konstruktor po klasie Animal
39  b = Bird(0.2)
40  print(b.m)
41  # obiekt tej klasy ma dodatkową metodę flights()
42  b.flights()
43
44  # klasa Mammal dziedziczy konstruktor po klasie Animal
45  m = Mammal(1000)
46  print(m.m)
47  # obiekt tej klasy ma dodatkową metodę runs()
48  m.runs()
```

Politechnika Wrocławska

Dziedziczenie

dziedziczenie01.py

```
33  # utworzenie obiektu klasy Animal
34  a = Animal(1.)
35  print(a.m)
36  a.moves()
37
38  # klasa Bird dziedzi
39  b = Bird(0.2)
40  print(b.m)
41  # obiekt tej klasy m
42  b.flights()
43
44  # klasa Mammal dzied
45  m = Mammal(1000)
46  print(m.m)
47  # obiekt tej klasy ma dodatkową metodę runs()
48  m.runs()
```

```
1.0
Zwierzęta się przemieszczają.
0.2
Ptaki latają.
1000
Ssaki biegają.
```

Politechnika Wrocławska

Dziedziczenie

- Przykład pokazuje tworzenie hierarchii klas
- Klasy pochodne przeciążają metodę klasy bazowej

- Pozwala to uzyskać polimorfizm metody moves(), której działanie zależy od klasy obiektu dla którego będzie wywołana

Politechnika Wrocławska

Dziedziczenie

dziedziczenie01.py dziedziczenie02.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Prosty przykład ilustrujący tworzenie hierarchii klas.
4  Klasy pochodne przeciążają metodę klasy bazowej.
5  """
6
7  # definicja klasy bazowej reprezentującej zwierzę
8  class Animal:
9      def __init__(self, m):
10         self.m     = m
11
12     # klasa implemenutuje metodę moves(),
13     # która będzie przeciążana przez klasy pochodne
14     def moves(self):
15         print("Zwierzęta się przemieszczają.")
```

Politechnika Wrocławska

Dziedziczenie

dziedziczenie01.py dziedziczenie02.py

```
17  class Bird(Animal):
18      # klasa Bird przeciąża metodę moves()
19      # pochodzącą z klasy bazowej
20      def moves(self):
21          print("Ptaki latają.")
22
23  class Mammal(Animal):
24      # klasa Mammal również przeciąża metodę moves()
25      # pochodzącą z klasy bazowej
26      def moves(self):
27          print("Ssaki biegają.")
28
29  # klasa Reptiles nie przeciąża metody moves()
30  class Reptiles(Animal):
31      pass
```

Politechnika Wrocławska

Dziedziczenie

dziedziczenie01.py dziedziczenie02.py

```
33  a = Animal(1.)
34  b = Bird(0.2)
35  m = Mammal(1000)
36  r = Reptiles(300)
37  # obiekty różnych klas zebranej na wspólnej liście
38  animals = [a, b, m, r]
39
40  # wywołanie metody moves() uruchomi różne wersje tej metody
41  # w zależności od klasy obiektu
42  for z in animals:
43      print(z.m)
44      z.moves()
```

Politechnika Wrocławska

Dziedziczenie

dziedziczenie01.py dziedziczenie02.py

```
33  a = Animal(1.)
34  b = Bird(0.2)
35  m = Mammal(1000)
36  r = Reptiles(300)
37  # obiekty różnych klas zebranej na wspólnej liście
38  animals = [a, b, m, r]
39
40  # wywołanie metody moves() uruchomi różne wersje tej metody
41  # w zależności od klasy obiektu
42  for z in animals:
```

wywołanie dla klasy Animal

wywołanie dla klasy Bird

wywołanie dla klasy Mammal

wywołanie dla klasy Reptile
(implementacja w klasie Animal)

```
1.0
Zwierzęta się przemieszczają.
0.2
Ptaki latają.
1000
Ssaki biegają.
300
Zwierzęta się przemieszczają.
```

Metody i atrybuty statyczne

- W pewnych sytuacjach pożądane jest, aby wszystkie instancje klasy współdzieliły pewne dane i/lub metody
- W pewnych sytuacjach pożądane może być, aby metody klasy były dostępne bezpośrednio z klasy (bez istniejących instancji)

Politechnika Wrocławska

Metody i atrybuty statyczne

- Przykładowo:
  - klasa jest wyposażona w licznik instancji – zmienną, która jest powiększana przy każdym wywołaniu konstruktora
- Istnieje kilka sposobów, aby zrealizować takie funkcjonalności

Politechnika Wrocławska

Metody i atrybuty statyczne

- Przykład wykorzystuje atrybut oraz metodę wspólne dla wszystkich obiektów klasy
- Metody nie można wywołać z konkretnej instancji

Politechnika Wrocławska

Metody i atrybuty statyczne

spam.py - 01 | spam.py - 02 | main.py - 02 | spam.py - 03 | main.py - 03

```python
# -*- coding: utf-8 -*-
"""
Przykład pokazuje wykorzystanie atrybutu i metody,
które są wspólne dla całej klasy.
"""

class Spam:

    # num_instances jest utworzone w klasie
    # a nie w instancji klasy
    num_instances = 0

    def __init__(self):
        # powiększenie licznika instancji
        # przez konstruktor
        Spam.num_instances += 1

    # metoda print_num_instances() nie przyjmuje
    # argumentu self
    def print_num_instances():
        print("Liczba utworzonych instancji:",
              Spam.num_instances)
```

Politechnika Wrocławska

Metody i atrybuty statyczne

spam.py - 01 | spam.py - 02 | main.py - 02 | spam.py - 03 | main.py - 03

```python
# -*- coding: utf-8 -*-
"""
Przykład pokazuje wykorzystanie atrybutu i metody,
które są wspólne dla całej klasy.
"""

class Spam:

    # num_instances jest utworzone w klasie
    # a nie w instancji klasy
    num_instances = 0

    def __init__(self):
        # powiększenie licznika instancji
        # przez konstruktor
        Spam.num_instances += 1

    # metoda print_num_instances() nie przyjmuje
    # argumentu self
    def print_num_instances():
        print("Liczba utworzonych instancji:",
              Spam.num_instances)
```

```python
def main():
    a = Spam()
    b = Spam()
    c = Spam()
    Spam.print_num_instances()
    # metody zdefiniowanej w ten sposób nie
    # można wywołać, na rzecz obiektu klasy.
    # a.print_num_instances()
    # Spam.print_num_instances(a)


if __name__ == "__main__":
    main()
```

Politechnika Wrocławska

Metody i atrybuty statyczne

spam.py - 01 | spam.py - 02 | main.py - 02 | spam.py - 03 | main.py - 03

```python
# -*- coding: utf-8 -*-
"""
Przykład pokazuje wykorzystanie atrybutu i metody,
które są wspólne dla całej klasy.
"""

class Spam:

    # num_instances jest utworzone w klasie
    # a nie w instancji klasy
    num_instances = 0

    def __init__(self):
        # powiększenie licznika instancji
        # przez konstruktor
        Spam.num_instances += 1

    # metoda print_num_instances() nie przyjmuje
    # argumentu self
    def print_num_instances():
        print("Liczba utworzonych instancji:",
              Spam.num_instances)
```

```python
def main():
    a = Spam()
    b = Spam()
    c = Spam()
    Spam.print_num_instances()
    # metody zdefiniowanej w ten sposób nie
    # można wywołać, na rzecz obiektu klasy.
    # a.print_num_instances()
    # Spam.print_num_instances(a)


if __name__ == "__main__":
    main()
```

```
Liczba utworzonych instancji: 3
```

Politechnika Wrocławska

Metody i atrybuty statyczne

- Przykład wykorzystuje atrybut utworzony w klasie oraz funkcję znajdującą się poza klasą, ale w jednym module
- Funkcja nie będzie dziedziczona

Politechnika Wrocławska

Metody i atrybuty statyczne

spam.py - 01 | spam.py - 02 | main.py - 02 | spam.py - 03 | main.py - 03

```python
# -*- coding: utf-8 -*-
"""
Przykład pokazuje wykorzystanie metody globalnej w module.
"""

def print_num_instances():
    print("Liczba utworzonych instancji:",
          Spam.num_instances)

class Spam:

    num_instances = 0

    def __init__(self):
        Spam.num_instances += 1
```

Politechnika Wrocławska

Metody i atrybuty statyczne

spam.py - 01 | spam.py - 02 | main.py - 02 | spam.py - 03 | main.py - 03

```python
# -*- coding: utf-8 -*-
"""
Przykład pokazuje wykorzystanie metody globalnej w module.
"""

def print_num_instances():
    print("Liczba utworzonych instancji:",
          Spam.num_instances)

class Spam:

    num_instances = 0

    def __init__(self):
        Spam.num_instances += 1
```

spam.py - 01 | spam.py - 02 | main.py - 02 | spam.py - 03 | main.py - 03

```python
# -*- coding: utf-8 -*-
"""
Wywołanie metody zdefiniowane w module "obok" klasy.
Taka metoda nie byłaby dziedziczona przez klasę pochodną.
"""

import spam

a = spam.Spam()
b = spam.Spam()
c = spam.Spam()
spam.print_num_instances()
print(spam.Spam.num_instances)
```

Metody i atrybuty statyczne

- Przykład wykorzystuje atrybut utworzony w klasie oraz metodę wywoływaną dla konkretnego obiektu klasy
- Metody nie można wywołać bez obiektu klasy

Politechnika Wrocławska

Metody i atrybuty statyczne

spam.py - 02 | main.py - 02 | spam.py - 03 | main.py - 03

```python
# -*- coding: utf-8 -*-
"""
Metoda print_num_instances() jest wywoływana
na rzecz konkretnej instancji klasy.
"""

class Spam:

    num_instances = 0

    def __init__(self):
        Spam.num_instances += 1

    def print_num_instances(self):
        print("Liczba utworzonych instancji:",
              Spam.num_instances)
```

Politechnika Wrocławska

Metody i atrybuty statyczne

spam.py - 02 | main.py - 02 | spam.py - 03 | main.py - 03

```
1   # -*- coding: utf-8 -*-
2   """
3   Metoda print_num_instances() jest wywoływana
4   na rzecz konkretnej instancji klasy.
5   """
6
7   class Spam:
8
9       num_instances = 0
10
11      def __init__(self):
12          Spam.num_instances +=
13
14      def print_num_instances(s
15          print("Liczba utworzo
16              Spam.num_instances)
```

spam.py - 02 | main.py - 02 | spam.py - 03 | main.py - 03

```
1   # -*- coding: utf-8 -*-
2   """
3   Przykład pokazuje definicję metody "statycznej" wywoływanej
4   na rzecz obiektu.
5   """
6
7   from spam import Spam
8
9   a, b, c = Spam(), Spam(), Spam()
10  a.print_num_instances()
11  Spam.print_num_instances(a)
12
13  # wywołanie konstruktura zmienia liczbę instancji
14  Spam().print_num_instances()
```

Politechnika Wrocławska

Metody i atrybuty statyczne

spam.py - 02 | main.py - 02 | spam.py - 03 | main.py - 03

```python
# -*- coding: utf-8 -*-
"""
Metoda print_num_instances() jest wywoływana
na rzecz konkretnej instancji klasy.
"""

class Spam:

    num_instances = 0

    def __init__(self):
        Spam.num_instances +=

    def print_num_instances(s
        print("Liczba utworzo
            Spam.num_instances)
```

```
Liczba utworzonych instancji: 3
Liczba utworzonych instancji: 3
Liczba utworzonych instancji: 4
```

spam.py - 02 | main.py - 02 | spam.py - 03 | main.py - 03

```python
# -*- coding: utf-8 -*-
"""
Przykład pokazuje definicję metody "statycznej" wywoływanej
na rzecz obiektu.
"""

from spam import Spam

a, b, c = Spam(), Spam(), Spam()
a.print_num_instances()
Spam.print_num_instances(a)

# wywołanie konstruktura zmienia liczbę instancji
Spam().print_num_instances()
```

Politechnika Wrocławska

Metody i atrybuty statyczne

- W klasie można zdefiniować metodę jako statyczną
- Można także zdefiniować metodę jako klasową
- Przykład pokazuje porównanie metod:
  - instancji,
  - statycznej,
  - klasowej.

Politechnika Wrocławska

Metody i atrybuty statyczne

methods.py | main.py

```python
# -*- coding: utf-8 -*-
"""
Klasa impementuje metody:
    - instancji,
    - statyczną,
    - klasową.
"""

class Methods:

    # definicja metody instancji
    def imeth(self, x):
        print([self, x])

    # definicja metody statycznej
    def smeth(x):
        print([x])

    # definicja metody klasowej
    def cmeth(cls, x):
        print([cls, x])

    # oznaczenie metody jako statycznej
    smeth = staticmethod(smeth)

    # oznaczenie meotdy jako klasowej
    cmeth = classmethod(cmeth)
```

Politechnika Wrocławska

Metody i atrybuty statyczne

methods.py | main.py

```python
# -*- coding: utf-8 -*-
"""
Klasa impementuje metody:
    - instancji,
    - statyczną,
    - klasową.
"""

class Methods:

    # definicja metody instancji
    def imeth(self, x):
        print([self, x])

    # definicja metody statycznej
    def smeth(x):
        print([x])

    # definicja metody klasowej
    def cmeth(cls, x):
        print([cls, x])

    # oznaczenie metody jako statycznej
    smeth = staticmethod(smeth)

    # oznaczenie meotdy jako klasowej
    cmeth = classmethod(cmeth)
```

methods.py | main.py

```python
# -*- coding: utf-8 -*-
"""
Przykład pokazuje porównanie metod:
    - instancji,
    - klasowej,
    - statycznej.
"""

from methods import Methods

obj = Methods()

obj.imeth(1)
Methods.imeth(obj, 2)

Methods.smeth(3)
obj.smeth(4)

Methods.cmeth(5)
obj.cmeth(6)
```

Politechnika Wrocławska

Metody i atrybuty statyczne

methods.py | main.py

```python
# -*- coding: utf-8 -*-
"""
Klasa impementuje metody:
    - instancji,
    - statyczną,
    - klasową.
"""

class Methods:

    # definicja metody instancji
    def imeth(self, x):
        print([self, x])

    # definicja metody statycznej
    def smeth(x):
        print([x])

    # definicja metody klasowej
    def cmeth(cls, x):
        print([cls, x])

    # oznaczenie metody jako statycznej
    smeth = staticmethod(smeth)

    # oznaczenie meotdy jako klasowej
    cmeth = classmethod(cmeth)
```

methods.py | main.py

```python
# -*- coding: utf-8 -*-
"""
Przykład pokazuje porównanie metod:
    - instancji,
    - klasowej,
    - statycznej.
"""

from methods import Methods

obj = Methods()

obj.imeth(1)
Methods.imeth(obj, 2)

Methods.smeth(3)
obj.smeth(4)

Methods.cmeth(5)
obj.cmeth(6)
```

```
[<methods.Methods object at 0x000002A3554A6A30>, 1]
[<methods.Methods object at 0x000002A3554A6A30>, 2]
[3]
[4]
[<class 'methods.Methods'>, 5]
[<class 'methods.Methods'>, 6]
```

Podsumowanie

- Dziedziczenie
- Wprowadzenie do zagadnienia atrybutów i metod statycznych