

Programowanie obiektowe

INP001045L

rok akademicki 2022/23

semestr zimowy

Laboratorium 12

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 220



Plan wykładu

- Dziedziczenie
- Wprowadzenie do zagadnienia atrybutów i metod statycznych



Dziedziczenie

- Mechanizm dziedziczenia pozwala uprościć proces tworzenia różnych klas, jeśli można dla nich wyróżnić pewne wspólne cechy
- Klasa pochodna (nazywana podklasą) jest tworzona na podstawie klasy bazowej (nadklasy)



Dziedziczenie

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



Dziedziczenie



dziedziczenie01.py ✕

```
1  # -*- coding: utf-8 -*-
2  """
3  Prosty przykład ilustrujący tworzenie hierarchii klas.
4  """
5
6  # definicja klasy bazowej reprezentującej zwierzę
7  class Animal:
8      # klasa implementuje konstruktor
9      # jedynym atrybutem obiektu jest m (masa)
10     def __init__(self, m):
11         self.m = m
12
13     # klasa implementuje metodę moves(),
14     # która wyświetla ogólną informację o zwierzętach
15     def moves(self):
16         print("Zwierzęta się przemieszczają.")
```



Dziedziczenie



dziedziczenie01.py ✕

```
18 # definicja klasy pochodnej
19 class Bird(Animal):
20     # klasa pochodna ma wszystkie cechy klasy bazowej,
21     # a dodatkowo implementuje metode 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 metode runs(),
29     # która wyświetla informację o ssakach
30     def runs(self):
31         print("Ssaki biegną.")
```



Dziedziczenie

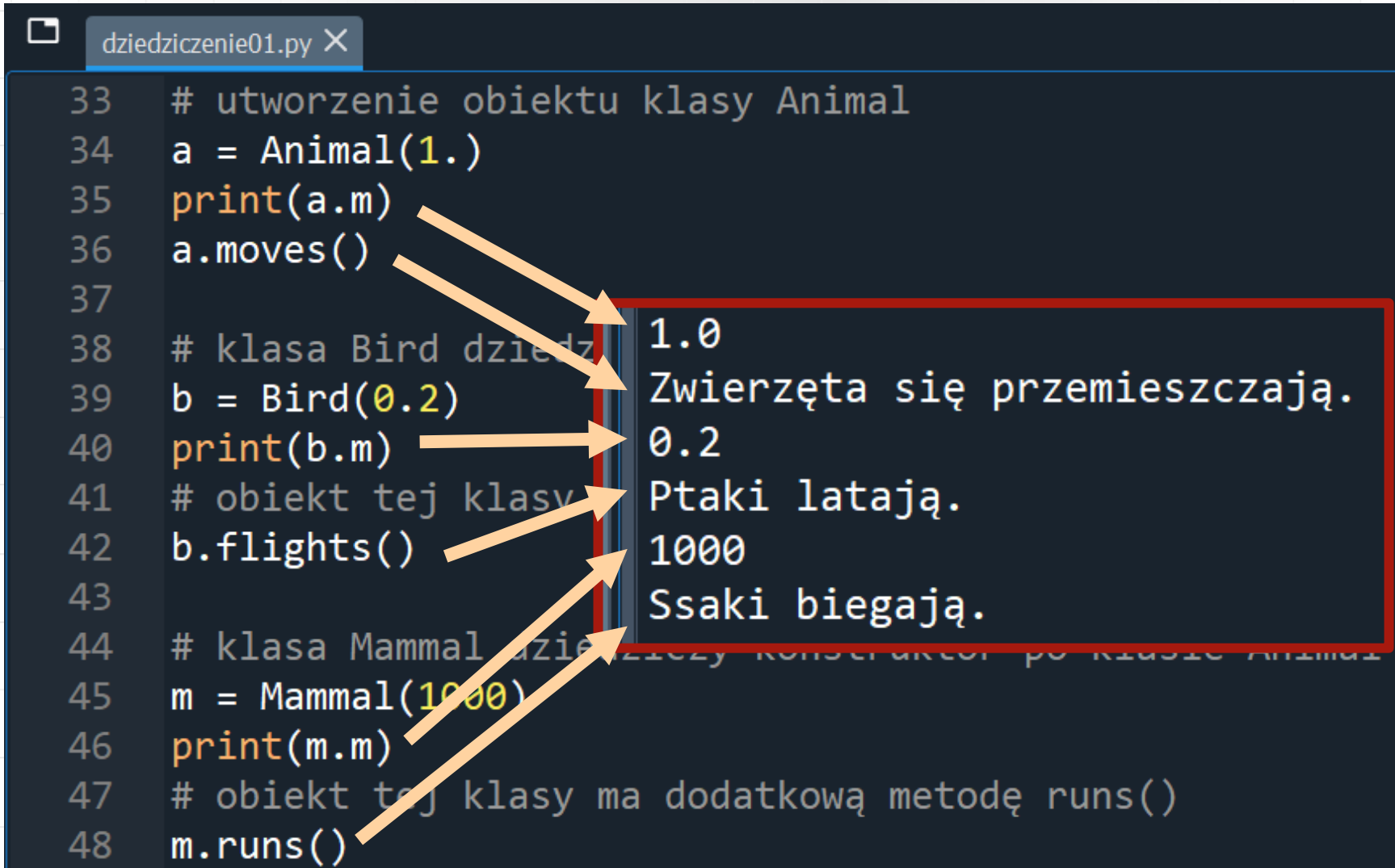


dziedziczenie01.py X

```
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()
```

Dziedziczenie

```
dziedziczenie01.py X
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
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()
```



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




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



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 implementuje metodę moves(),
13         # która będzie przeciążana przez klasy pochodne
14         def moves(self):
15             print("Zwierzęta się przemieszczają.")
```



Dziedziczenie



dziedziczenie01.py X

dziedziczenie02.py X

```
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
```



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()
```



Dziedziczenie

```
dziedziczenie01.py x dziedziczenie02.py x
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
```

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



Metody i atrybuty statyczne

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



Metody i atrybuty statyczne

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



Metody i atrybuty statyczne

```
spam.py - 01 X spam.py - 02 X main.py - 02 X spam.py - 03 X main.py - 03 X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazuje wykorzystanie atrybutu i metody,
4  które są wspólne dla całej klasy.
5  """
6
7  class Spam:
8
9      # num_instances jest utworzone w klasie
10     # a nie w instancji klasy
11     num_instances = 0
12
13     def __init__(self):
14         # powiększenie licznika instancji
15         # przez konstruktor
16         Spam.num_instances += 1
17
18     # metoda print_num_instances() nie przyjmuje
19     # argumentu self
20     def print_num_instances():
21         print("Liczba utworzonych instancji:",
22               Spam.num_instances)
```



Metody i atrybuty statyczne

```
spam.py - 01 X spam.py - 02 X main.py - 02 X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazuje wykorzystanie
4  które są wspólne dla całej klasy
5  """
6
7  class Spam:
8
9      # num_instances jest atrybutem statycznym
10     # a nie w instancji
11     num_instances = 0
12
13     def __init__(self):
14         # powiększenie licznika instancji
15         # przez konstruktor
16         Spam.num_instances += 1
17
18     # metoda print_num_instances() nie przyjmuje
19     # argumentu self
20     def print_num_instances():
21         print("Liczba utworzonych instancji:",
22               Spam.num_instances)
23
24     def main():
25         a = Spam()
26         b = Spam()
27         c = Spam()
28         Spam.print_num_instances()
29         # metody zdefiniowanej w ten sposób nie
30         # można wywołać, na rzecz obiektu klasy.
31         # a.print_num_instances()
32         # Spam.print_num_instances(a)
33
34
35     if __name__ == "__main__":
36         main()
```



Metody i atrybuty statyczne

```
spam.py - 01 X spam.py - 02 X main.py - 02 X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazuje wykorzystanie atrybutów statycznych
4  które są wspólne dla całej klasy.
5  """
6
7  class Spam:
8
9      # num_instances jest atrybutem statycznym
10     # a nie w instancji
11     num_instances = 0
12
13     def __init__(self):
14         # powiększenie licznika instancji
15         # przez konstruktor
16         Spam.num_instances += 1
17
18     # metoda print_num_instances() nie przyjmuje
19     # argumentu self
20     def print_num_instances():
21         print("Liczba utworzonych instancji:",
22               Spam.num_instances)
23
24     def main():
25         a = Spam()
26         b = Spam()
27         c = Spam()
28         Spam.print_num_instances()
29         # Liczba utworzonych instancji: 3
30         # można wywołać, na rzecz obiektu klasy.
31         # a.print_num_instances()
32         # Spam.print_num_instances(a)
33
34
35     if __name__ == "__main__":
36         main()
```



Metody i atrybuty statyczne

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



Metody i atrybuty statyczne

```
spam.py - 01 X spam.py - 02 X main.py - 02 X spam.py - 03 X main.py - 03 X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazuje wykorzystanie metody globalnej w module.
4  """
5
6  def print_num_instances():
7      print("Liczba utworzonych instancji:",
8            Spam.num_instances)
9
10 class Spam:
11
12     num_instances = 0
13
14     def __init__(self):
15         Spam.num_instances += 1
```



Metody i atrybuty statyczne

```
spam.py - 01 X spam.py - 02 X main.py - 02 X spam.py - 03 X main.py - 03 X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazuje wykorzystanie metody globalnej w module.
4  """
5
6  def print_num_instances():
7      ...
8
9
10 cla
11
12
13
14
15
```

```
spam.py - 01 X spam.py - 02 X main.py - 02 X spam.py - 03 X main.py - 03 X
1  # -*- coding: utf-8 -*-
2  """
3  Wywołanie metody zdefiniowane w module "obok" klasy.
4  Taka metoda nie byłaby dziedziczona przez klasę pochodną.
5  """
6
7  import spam
8
9  a = spam.Spam()
10 b = spam.Spam()
11 c = spam.Spam()
12 spam.print_num_instances()
13 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



Metody i atrybuty statyczne

```
spam.py - 02 X main.py - 02 X spam.py - 03 X main.py - 03 X
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 += 1
13
14     def print_num_instances(self):
15         print("Liczba utworzonych instancji:",
16               Spam.num_instances)
```




Metody i atrybuty statyczne

```
spam.py - 02 X main.py - 02 X spam.py - 03 X main.py - 03 X
1 # -*- coding: utf-8 -*-
2 """
3 Metoda print_num_instances() jest wywoływana
4 na rzecz konkretnej instancji klasy.
5 """
6
7 c] spam.py - 02 X main.py - 02 X spam.py - 03 X main.py - 03 X
8
9
10 1 # -*- coding: utf-8 -*-
11 2 """
12 3 Przykład pokazuje definicję metody "statycznej" wywoływanej
13 4 na rzecz obiektu.
14 5 """
15 6
16 7 from spam import Spam
17 8
18 9 a, b, c = Spam(), Spam(), Spam()
19 10 a.print_num_instances()
20 11 Spam.print_num_instances(a)
21 12
22 13 # wywołanie konstruktora zmienia liczbę instancji
23 14 Spam().print_num_instances()
```



Metody i atrybuty statyczne

```
spam.py - 02 X main.py - 02 X spam.py - 03 X main.py - 03 X
1  # -*- coding: utf-8 -*-
2  """
3  Metoda print_num_instances() jest
4  na rzecz konkretnej instancji kla
5  """
6
7  c] spam.py - 02 X main.py - 02 X spam.py - 03 X main.py - 03 X
8
9
10 1  # -*- coding: utf-8 -*-
11 2  """
12 3  Przykład pokazuje definicję metody "statycznej" wywoływanej
13 4  na rzecz obiektu.
14 5  """
15 6
16 7  from spam import Spam
17 8
18 9  a, b, c = Spam(), Spam(), Spam()
19 10 a.print_num_instances()
20 11 Spam.print_num_instances(a)
21 12
22 13 # wywołanie konstruktora zmienia liczbę instancji
23 14 Spam().print_num_instances()
```

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



Metody i atrybuty statyczne

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



Metody i atrybuty statyczne

```
methods.py × main.py ×
1  # -*- coding: utf-8 -*-
2  """
3  Klasa impementuje metody:
4  - instancji,
5  - statyczna,
6  - klasowa.
7  """
8
9  class Methods:
10
11     # definicja metody instancji
12     def imeth(self, x):
13         print([self, x])
14
15     # definicja metody statycznej
16     def smeth(x):
17         print([x])
18
19     # definicja metody klasowej
20     def cmeth(cls, x):
21         print([cls, x])
22
23     # oznaczenie metody jako statycznej
24     smeth = staticmethod(smeth)
25
26     # oznaczenie meotdy jako klasowej
27     cmeth = classmethod(cmeth)
```



Metody i atrybuty statyczne

```
methods.py X main.py X
1  # -*- coding: utf-8 -*-
2  """
3  Klasa impementuje metody:
4  - instancji,
5  - statyczna,
6  - klasowa.
7  """
8
9  class Methods:
10
11     # definicja metody instancji
12     def imeth(self, x):
13         print([self, x])
14
15     # definicja metody statycznej
16     def smeth(x):
17         print([x])
18
19     # definicja metody klasowej
20     def cmeth(cls, x):
21         print([cls, x])
22
23     # oznaczenie metody jako statycznej
24     smeth = staticmethod(smeth)
25
26     # oznaczenie meotdy jako klasowej
27     cmeth = classmethod(cmeth)
```

```
methods.py X main.py X
1  # -*- coding: utf-8 -*-
2  """
3  Przykład pokazuje porównanie metod:
4  - instancji,
5  - klasowej,
6  - statycznej.
7  """
8
9  from methods import Methods
10
11  obj = Methods()
12
13  obj.imeth(1)
14  Methods.imeth(obj, 2)
15
16  Methods.smeth(3)
17  obj.smeth(4)
18
19  Methods.cmeth(5)
20  obj.cmeth(6)
```



Metody i atrybuty statyczne

```
methods.py X main.py X
1 # -*- coding: utf-8 -*-
2 """
3 Klasa impementuje metody:
4 - instancji,
5 - statyczna,
6 - klasowa.
7 """
8
9 class Methods:
10
11     # definicja metody instancji
12     def imeth(self, x):
13         print([self, x])
14
15     # definicja metody statycznej
16     def smeth(x):
17         print([x])
18
19     # definicja metody klasowej
20     def cmeth(cls, x):
21         print([cls, x])
22
23     # oznaczenie metody j
24     smeth = staticmethod(
25
26     # oznaczenie meotdy j
27     cmeth = classmethod(c
```

```
methods.py X main.py X
1 # -*- coding: utf-8 -*-
2 """
3 Przykład pokazuje porównanie metod:
4 - instancji,
5 - klasowej,
6 - statycznej.
7 """
8
9 from methods import Methods
10
11 obj = Methods()
12
13 obj.imeth(1)
14 Methods.imeth(obj, 2)
15
16 Methods.smeth(3)
17 obj.smeth(4)
18
19 Methods.cmeth(5)
20 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