Objektovo orientované programovanie - *pokročilí*

Učiteľ:
Ing. Jozef Wagner PhD.

Učebnica:
https://oop.wagjo.com/

OPGP
Pokročilí 12

1. Dedičnosť
2. super()
3. Viacnásobná dedičnosť
4. Duck typing

Dedičnosť

Dedičnosť umožňuje vytvoriť novú triedu (podtriedu), ktorá **preberá vlastnosti a metódy** inej triedy (nadtriedy).

Užitočné metódy:

- `issubclass(Student, Osoba)`
- `isinstance(d, Student)`
- `isinstance(d, Osoba)` – aj rodičovské triedy

Dedičnosť

```
class Osoba:
    def __init__(self, meno):
        self.meno = meno
    def pozdrav(self):
        print(f"Ahoj, som {self.meno}")


class Student(Osoba):
    def __init__(self, meno, skola):
        super().__init__(meno)  # voláme konštruktor rodiča
        self.skola = skola

    # prepísanie (override) metódy
    def pozdrav(self):
        print(f"Som {self.meno} a chodim na {self.skola}")
```

super()

Pomocou funkcie `super()` vieme pristupovať k objektu rodiča (a volať jeho metódy a konštruktory a používať jeho atribúty).

Využitie:

- **volanie rodičovského konštruktora** - aby sa dosiahla správna inicializácia atribútov
- **volanie prepísanej (prekrytej) metódy** - ak z nejakého dôvodu metóda v podtriede potrebuje zavolať prekrytú metódu

super()

```
class A:
    def __init__(self, a=None, **kwargs):
        super().__init__(**kwargs)  # vždy ako prvé!
        self.a = a
        print(f"A: a = {a}")

class B:
    def __init__(self, b=None, **kwargs):
        super().__init__(**kwargs)
        self.b = b
        print(f"B: b = {b}")
```

super()

```
class C:
    def __init__(self, c=None, **kwargs):
        super().__init__(**kwargs)
        self.c = c
        print(f"C: c = {c}")

# Akýkoľvek poradie dedičnosti funguje!
class MojaTrieda(A, B, C):
    def __init__(self, x=None, **kwargs):
        super().__init__(**kwargs) # inicializuje A, potom B a C
        self.x = x
        print(f"MojaTrieda: x = {x}")
```

Viacnásobná dedičnosť

Na rozdiel od Javy sa v Pythone dá použiť viacnásobná dedičnosť

```
class A:
    def akcia(self):
        print("A")

class B(A):
    def akcia(self):
        print("B")
        super().akcia()
```

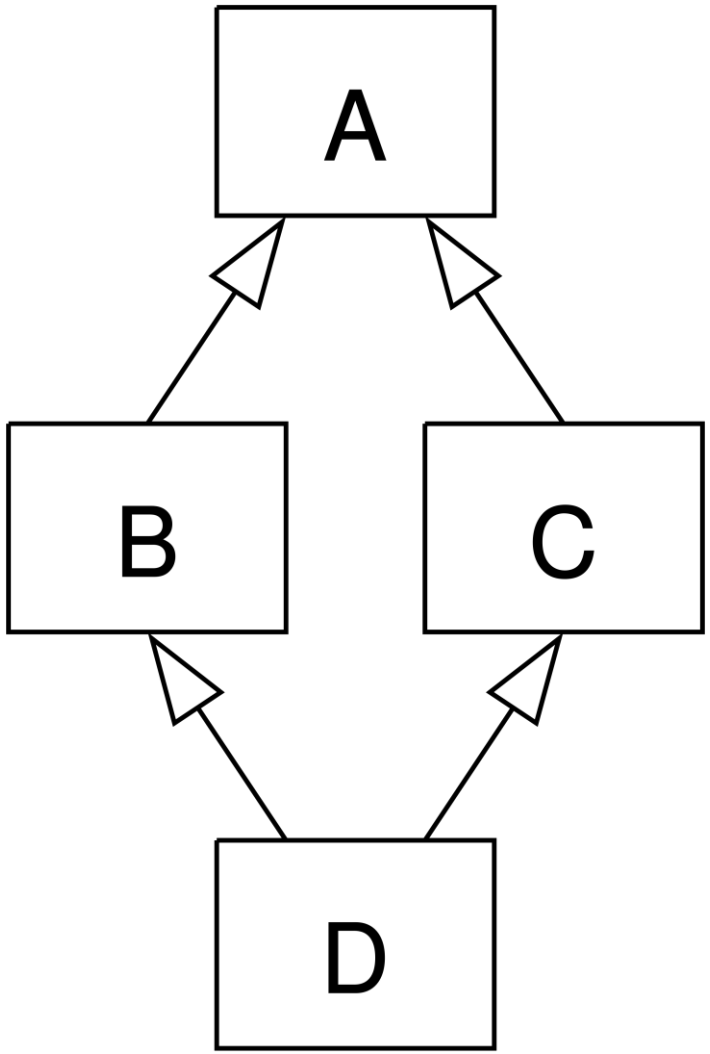
```
class C(A):
    def akcia(self):
        print("C")
        super().akcia()

class D(B, C):  # D dedí od B aj C
    def akcia(self):
        print("D")
        super().akcia()
```

Viacnásobná dedičnosť

Diamantový problém:
niektoré triedy môžu byť dedené viackrát. Problém pri prekrytých metódach.

Python na určenie, ktorá metóda sa zavolá použije tzv. algoritmus **MRO** (**Method Resolution Order**)



Viacnásobná dedičnosť

```
print(D.mro())
# [<class 'D'>, <class 'B'>,
   <class 'C'>, <class 'A‘>,
   <class 'object'>]

d = D()
d.akcia()

# Výstup:
# D
# B
# C
# A
```

Duck typing

„Ak to chodí ako kačka a kváka ako kačka, tak to je kačka.“

Pythonu nezáleží na type objektu, ale či má požadované metódy

```
class Pes:
    def zvuk(self):
        return "Haf haf!"

class Macka:
    def zvuk(self):
        return "Mňau!"

class Kacer:
    def zvuk(self):
        return "Kvák!"
```

```
def zvuk_zvierata(zviera):
    print(zviera.zvuk())

for z in [Pes(), Macka(), Kacer()]:
    zvuk_zvierata(z)

# Haf haf!
# Mňau!
# Kvák!
```

Hra adventúra

Dnes budeme vylepšovať hru adventura, ktorú sme si programovali na minulom cvičení. Zameriame sa na nasledovné funkcionality:

- inventár hráča
- príkaz na vloženie veci do inventára
- použitie veci z inventára