Programovanie v jazyku Python

Princípy objektovo orientovaného programovania
prednáška 7

Katedra kybernetiky a umelej inteligencie
Technická univerzita v Košiciach
Ing. Ján Magyar, PhD.

Princípy OOP

1. abstrakcia
2. enkapsulácia
3. dedenie
4. polymorfizmus

Abstrakcia

- pomocou abstrakcie skryjeme implementačné detaily o funkcionalite
- každá trieda by mala poskytovať iba API - súbor metód pre prácu s vnútornými premennými
- cieľom je, aby ďalšie objekty a programátor sa nezaoberali tým, ako presne funguje daná metóda, ale iba tým, čo robí
- pre vysokú mieru abstrakcie je nevyhnutná správna forma dokumentácie, najmä ak metóda má vedľajšie účinky

Abstrakcia v Pythone

1. ak používame triedy, tak neriešime, ako implementujú funkcionalitu
   - spoliehame sa na dokumentáciu a na autora daného kódu
2. keďže trieda definuje abstraktný dátový typ, nezaoberáme sa ani vnútornou reprezentáciou údajov
   - napr. hašovacia tabuľka môže byť reprezentovaná ako slovník alebo ako zoznam zoznamov
   - vnímame iba vonkajší kontext triedy a nie vnútorné detaily

Abstrakcia - ukážka

Máme už známu triedu hašovacej tabuľky z cvičenia. Pre reprezentáciu hašovacej tabuľky používame dictionary. Vzhľadom na to, že hašovacia funkcia vráti zvyšok po delení 6, možné kľúče sú 0, 1, 2, 3, 4, 5.

Upravíme triedu tak, že reprezentáciu vymeníme na zoznam zoznamov:
- pomocou prvého indexu vyberieme zoznam pod daným kľúčom
- jednotlivé hodnoty budú uložené pod druhým indexom vo vybranom zozname

Enkapsulácia

- kým abstrakcia skryje implementačné detaily, cieľom enkapsulácie je skryť vnútorný stav objektu
- vnútorný stav je definovaný ako súbor hodnôt vo vnútorných premenných
- enkapsulácia definuje spôsob ako môžeme narábať s objektom z danej triedy

Enkapsulácia v Jave/C# vs. v Pythone

- objektovo orientované jazyky založené na C podporujú enkapsuláciu implicitne
- pre každý atribút vieme definovať viditeľnosť cez kľúčové slová: `public, private, protected, package-private`
- Python tieto kľúčové slová nemá, nemá ani balíky, programátor musí implementovať enkapsuláciu explicitne
- best practice: v rámci triedy pristupovať k vnútorným premenným priamo, mimo triedy cez pomocné metódy

Privátne atribúty v Pythone

- v triede môžeme zadefinovať atribúty ako privátne pomocou znakov __

```
class Product:
    def __init__(self):
        self.__price = 1000

t = Product()
t.__price = 500
```

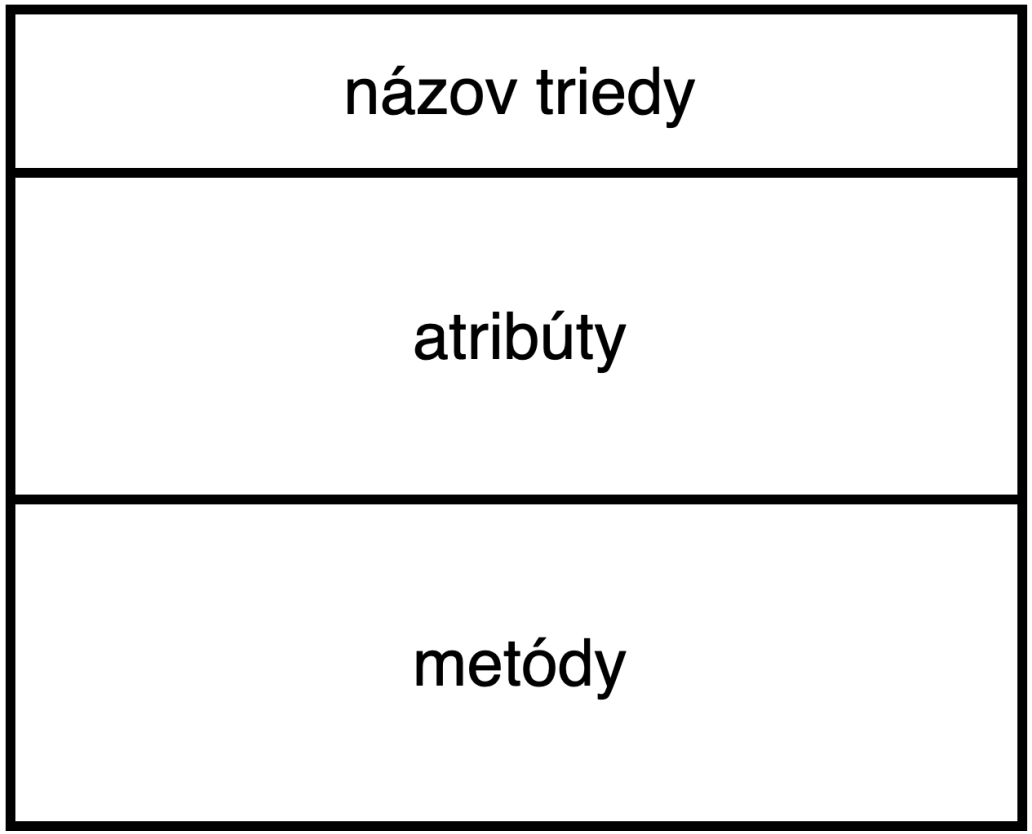
Stav objektu

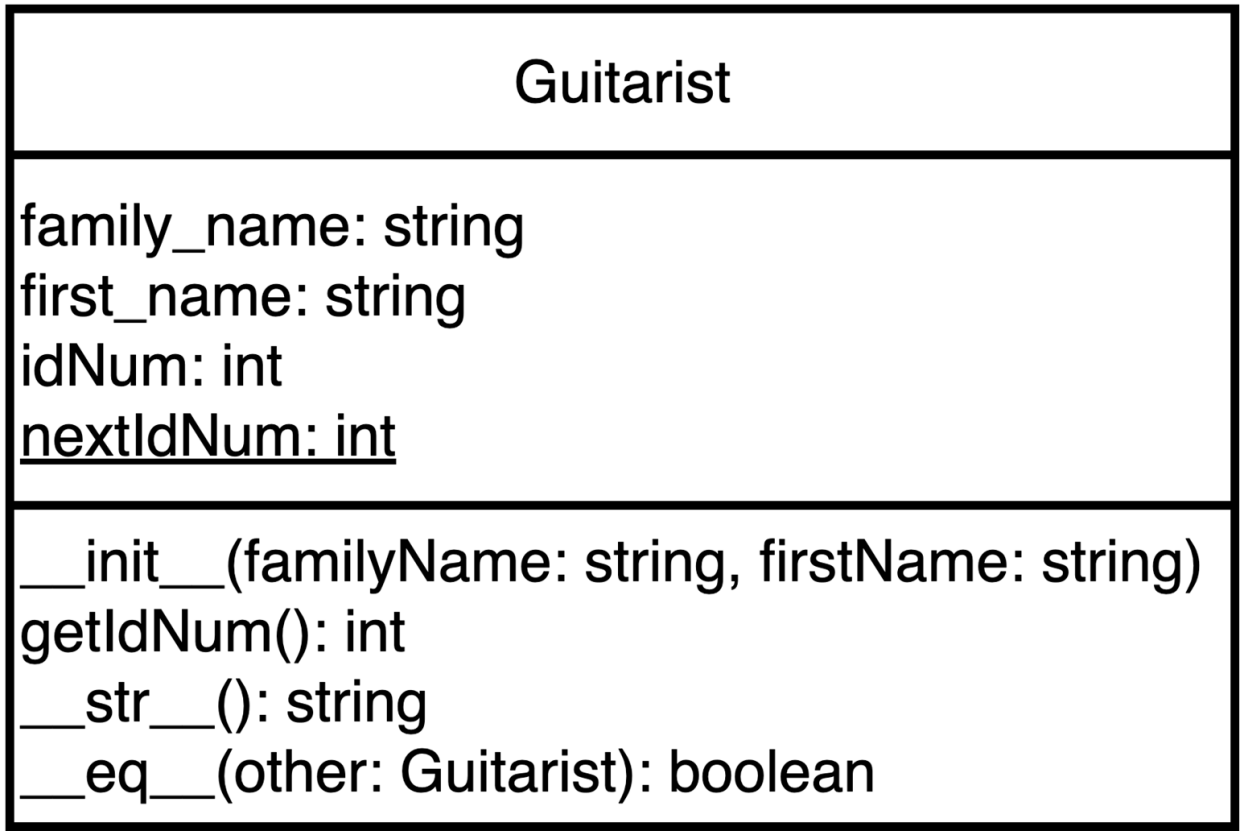
- stav objektu je súhrn hodnôt všetkých premenných objektu
- dva základné typy premenných:
  - členské premenné – jedinečné pre objekty
  - premenné triedy – jedinečné pre triedu
- stav objektu sa počas behu programu mení

Premenné tried

- premenné typu `self.name` sú špecifické pre každú inštanciu
- Python umožňuje používanie premenných, ktoré zdieľajú všetky inštancie danej triedy
- definujeme ich mimo konštruktora
- najčastejšie aplikácie
  - jedinečný index (práca s databázami)
  - počítadlo
  - pomocná premenná pre niektoré návrhové vzory (najmä singleton)

Class diagram





Enkapsulácia

- v Pythone nemáme príznaky, definujeme to priamo v názve premennej
  - `idNum: int` - public premenná
  - `_idNum: int` - private premenná

- premenné triedy sú podčiarknuté
  - `nextIdNum: int`

Dedenie

- dedenie znamená, že trieda “dedí” časť funkcionality od inej triedy
- trieda, ktorá dedí je **podtrieda**; trieda, od ktorej dedí je **nadtrieda**
- pomocou dedenia vieme vytvoriť hierarchiu, kde vytvoríme stále bližšiu špecifikáciu tried - podtriedy budú presnejšie definované verzie nadtriedy
- v Python 2 bolo možné definovať triedy bez nadtried, v Python 3 defaultná nadtrieda je `object`
- Python podporuje viacnásobné dedenie

Definícia dedenia v Pythone

```
class Person:
    def __init__(self, name):
        self.name = name


class Student(Person):
    def __init__(self, name, year):
        self.name = name
        self.year = year
```

Volanie metód nadtriedy

- pre podporu polymorfizmu a efektívnu prácu s triedami by sme mali použiť čo najviac už definovanú funkcionalitu v nadtriedach
- kľúčové slovo `super()`

```
class Person:
    def __init__(self, name):
        print('setting name in Person')
        self.name = name

class Student(Person):
    def __init__(self, name, year):
        super().__init__(name)
        self.year = year

janko_hrasko = Student("Janko Hrasko", 1)
```

Viacnásobné dedenie v Pythone

```
class Person: pass
class Worker(Person): pass
class Boss(Person): pass

class Manager(Worker, Boss): pass
class Secretary(Worker): pass
class BoardMember(Boss): pass

class CEO(Manager, BoardMember): pass
```

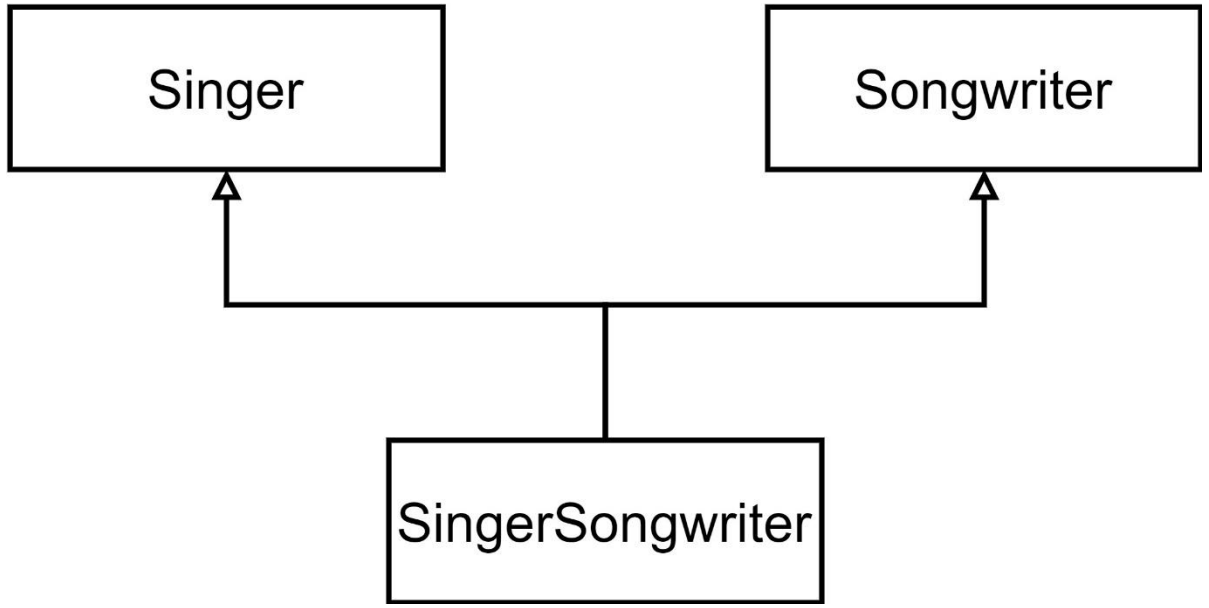
Problém viacnásobného dedenia

v triede `Singer`:

```
sings = True
writes = False
```

v triede `Songwriter`:

```
sings = False
writes = True
```

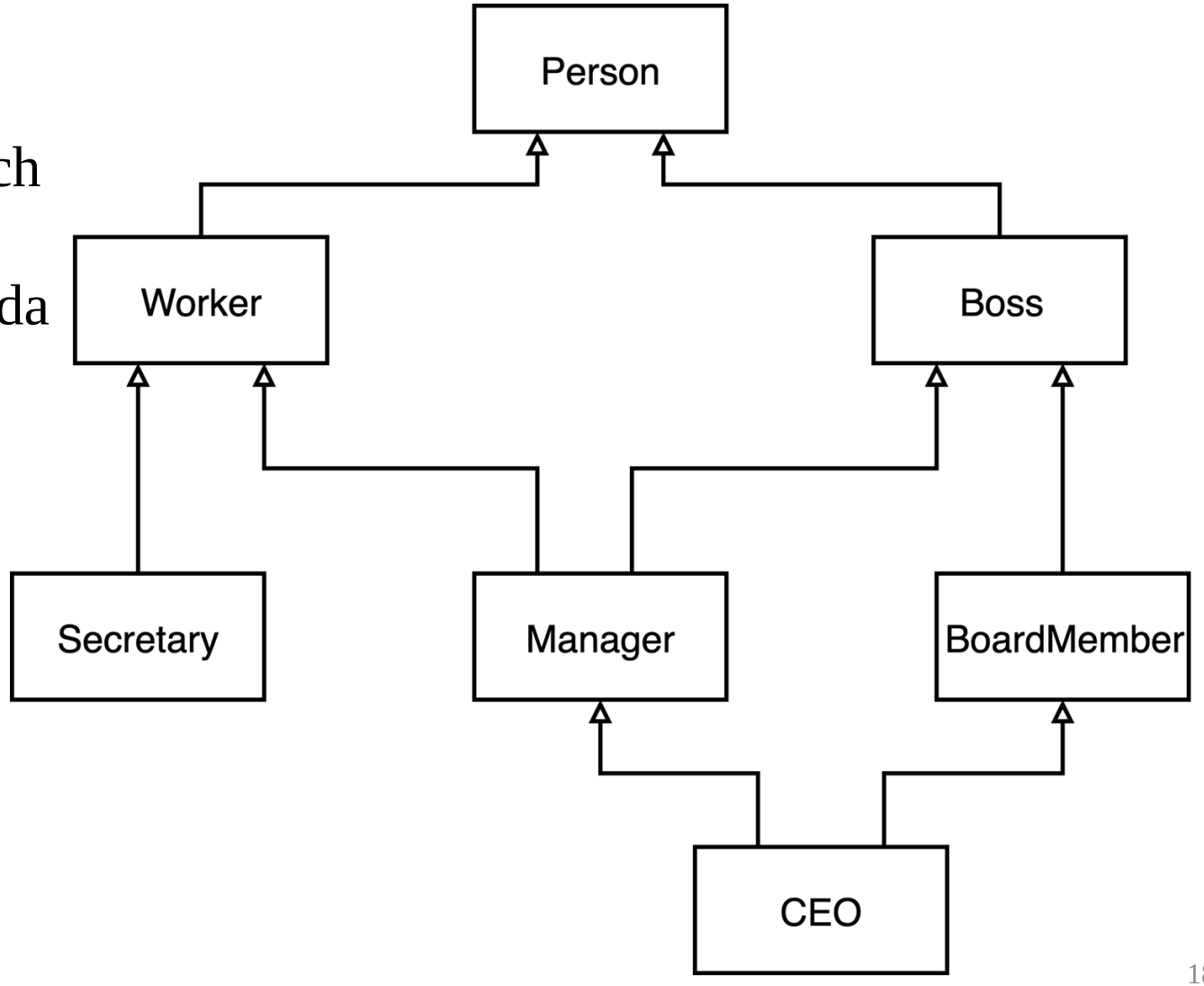


Aké hodnoty zdedí trieda `SingerSongwriter`?

Name resolution pri viacnásobnom dedení

1. vyhľadávanie v aktuálnej triede
2. vyhľadávanie do hĺbky v nadtriedach a zľava doprava
3. ako posledná sa berie do úvahy trieda `object`

- poradie vieme zistiť pomocou premennej `__mro__` alebo metódou `mro()`



Polymorfizmus

- polymorfizmus nám umožňuje použiť rovnaké rozhranie pre rôzne dátové typy, medzi ktorými existuje dedenie
- reálne to znamená, že s objektmi z podtriedy vieme pracovať rovnakým spôsobom ako s objektmi z nadtriedy
- keďže Python je dynamicky typovaný jazyk, nie je až taký výrazný

Ukážka - reprezentácia bodov v priestore

v dvojrozmernom priestore vieme každý bod reprezentovať dvoma spôsobmi:

1. zápis pomocou karteziánskych súradníc - hodnoty x a y
2. zápis pomocou polárnych súradníc - vzdialenosť od bodu [0, 0] a uhol spojnice bodu a osi

Môžeme vytvoriť reprezentáciu bodu, ktorá je nezávislá od zápisu?

Zhrnutie

- abstrakcia
- enkapsulácia
- stav objektu
- členské premenné a premenné tried
- dedenie
- polymorfizmus