Uvod u programski jezik Pajton - 6. deo

Uvod u merno-informacione sisteme

Destruktor

- Destruktori su metode koje se pozivaju prilikom uništavanja objekata
- Uništavanje se može dogoditi:
  1. Automatski - kada objekat prestane biti dostupan
  2. Eksplicitno - kada pozovemo naredbu *del* nad objektom
- Python ima automatsko upravljanje memorijom pomoću sakupljača smeća (*garbage collector*)
- Objekti brišu automatski kada nisu više potrebni
- Ako postoji potreba da se manuelno obrišu onda se može eksplicitno pozvati destruktor

Destruktor

```
1  class Knjiga:
2      def __init__(self, naslov):
3          self.naslov = naslov
4          print(f"Napravljenja je knjiga {self.naslov}")
5
6      def __del__(self):
7          print(f"Knjiga {self.naslov} je unistena")
8
9  knjiga1 = Knjiga("Pajton za pocetnike")
10 knjiga2 = Knjiga("Uvod u Pajton")
11 print(knjiga1.naslov)#Pajton za pocetnike
12 del knjiga1
13 print(knjiga1.naslov)#NameError: name 'knjiga1' is not
       defined
```

Problem enkaspulacije u Pajtonu

- Prvi problem: *protected* atributu se može pristupiti i izvan klase
- Zbog toga se dopisivanje _ ispred atributa koristi samo kao informativna stvar
- Drugi problem: *private* atributu se takođe može pristupiti
- Pristup atributu se vrši *imeObjekta._ImeKlase__privatniAtribut*
- Ovaj način čuvanja atributa se naziva *mangling*, gde se pre imena stavlja ime klase
- Iako Pajton podržava enkaspulaciju, ona je više na nivou konvencije!

Problem enkaspulacije u Pajtonu

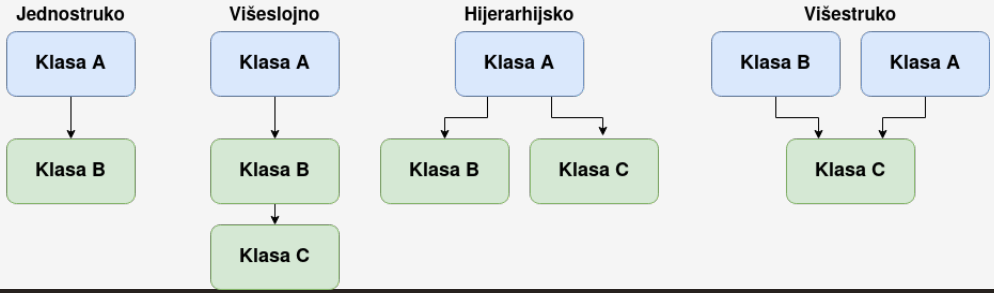
```
1  class BankovniRacun:
2      def __init__(self, broj_racuna, stanje):
3          self.__broj_racuna = broj_racuna
4          self._stanje = stanje
5
6      def get_broj_racuna(self):
7          return self.__broj_racuna
8
9  racun = BankovniRacun('1111-2222-3333-4444',25000)
10 print(racun.get_broj_racuna())#'1111-2222-3333-4444'
11 print(racun._stanje)#25000
12
13 racun._BankovniRacun__broj_racuna = '4444-3333-2222-1111'
14 print(racun.get_broj_racuna())#'4444-3333-2222-1111'
```

Nasleđivanje

- Nasleđivanje predstavlja mehanizam da se na osnovu drugih klasa izradi klasa, preuzimanjem već implementiranih funkcionalnosti
- Klasa koja od koje se nasleđuju podaci naziva se nadklasa, osnovna ili roditeljska klasa
- Klasa koja se realizuje nasleđivanjem naziva se izvedena podklasa, izvedena ili dete klasa
- Podklase nasleđuju atribute i metode nadklase, a mogu ih nadjačati, proširiti ili promeniti
- Nasleđivanje ostvaruje korišćenjem ključne reči class, navođenjem imena klase i imena roditeljske klase u zagradi
- Pristup metodama iz podklase vrši se korišćenjem *super* funkcije ili korišćenjem imena klase

Vrste nasleđivanja u Pajtonu

1. Jednostruko nasleđivanje - klasa može naslediti svojstva i metode samo od jedne nadklase
2. Višeslojno nasleđivanje - klasa nasleđuje svojstva i metode od klase koja je sama po sebi naslednik
3. Hijerarhijsko nasleđivanje - više od jedne klase nasleđuje se od jedne iste nadklase
4. Višestruko nasleđivanje - klasa može naslediti svojstva i metode od više od jedne nadklase



Jednostruko nasleđivanje

```
1  class Zivotinja:
2      def __init__(self, ime):
3          self.ime = ime
4      def jedi(self):
5          print(f'Zivotinja {self.ime} jede')
6      def kreci_se(self):
7          print(f'Zivotinja {self.ime} se krece')
8  class Riba(Zivotinja):
9      def __init__(self, ime):
10         super().__init__(ime)
11     def kreci_se(self):
12         print(f'{self.ime} pliva')
13
14 riba = Riba('Orada')
15 riba.kreci_se()#Orada pliva
16 riba.jedi()#Zivotinja Orada jede
```

Višeslojno nasleđivanje

```
1  class Ajkula(Riba):
2      def __init__(self, ime):
3          super().__init__(ime)
4
5      def napadni(self):
6          print(f'{self.ime} napada')
7
8  ajkula = Ajkula('Plava ajkula')
9  ajkula.kreci_se()#Plava ajkula pliva
10 ajkula.jedi()#Zivotinja Plava ajkula jede
11 ajkula.napadni()#Plava ajkula napada
```

Hijerarhijsko nasleđivanje

```
1  class Sisar(Zivotinja):
2      def __init__(self, ime):
3          super().__init__(ime)
4      def kreci_se(self):
5          print('Sisari se brze krecu')
6      def hranjenje_mlekom(self):
7          print('Sisar hrani mladunce')
8
9  riba = Riba('Orada')
10 sisar = Sisar('Konj')
11 riba.kreci_se()#Orada pliva
12 riba.jedi()#Zivotinja Orada jede
13 sisar.kreci_se()#Sisari se brze krecu
14 sisar.hranjenje_mlekom()#Sisar hrani mladunce
15 sisar.jedi()#Zivotinja Konj jede
```

Višestruko nasleđivanje

```
1  class Akrobata:
2      def __init__(self, ime, vestina):
3          self.ime = ime
4          self.vestina = vestina
5
6      def izvedi_akrobaciju(self):
7          print(f'{self.ime} izvodi {self.vestina}')
8
9  class Delfin(Riba, Akrobata):
10     def __init__(self, ime, vestina):
11         Riba.__init__(self, ime)
12         Akrobata.__init__(self, ime, vestina)
13
14     def igraj_se(self):
15         print(f'{self.ime} se igra')
```

Višestruko nasleđivanje - nastavak

```
1 delfin = Delfin('Delfi', 'skok')
2
3 delfin.igraj_se()#Delfi se igra
4 delfin.kreci_se()#Delfi pliva
5 delfin.izvedi_akrobaciju()#Delfi izvodi skok
6 delfin.jedi()#Zivotinja Delfi jede
```

Problem dijamantskog nasleđivanja

- Problem dijamantskog nasleđivanja nastaje kada dve klase nasleđuju jednu roditeljsku, a zatim se iz njih višestrukim nasleđivanjem izvodi sledeća klasa

```
1  class Delfin(Riba, Sisar):
2      def __init__(self, ime):
3          Riba.__init__(self, ime)
4          Sisar.__init__(self, ime)
5      def igraj_se(self):
6          print(f'{self.ime} se igra')
7
8  delfin = Delfin('Delfi')
9  delfin.igraj_se()
10 delfin.jedi()
11 delfin.kreci_se()#PROBLEM!!!
```

- I klasa Sisar i klasa Riba nadjačavaju metodu kreći se. Koja metoda će se pozvati?

Problem dijamantskog nasleđivanja

- U Pajtonu je ovo realizovano pozivom atributa *__mro__* nad željenom klasom
- MRO (*Method Resolution Order*) predstavlja algoritam koji se koristi da bi se odredio redosled poziva metoda koje će se izvršiti kada se pozove metoda koja je definisana u više klasa

```
1 print(Delfin.__mro__)
2 #(<class '__main__.Delfin'>,
3 # <class '__main__.Riba'>,
4 # <class '__main__.Sisar'>,
5 # <class '__main__.Zivotinja'>,
6 # <class 'object'>)
```

Višestruko nasleđivanje i *super*

```
1  class A:
2      def __init__(self):
3          print('Konstruktor A')
4  class B(A):
5      def __init__(self):
6          print('Konstruktor B')
7          super().__init__()
8  class C(A):
9      def __init__(self):
10         print('Konstruktor C')
11         super().__init__()
12 class D(B, C):
13     def __init__(self):
14         print('Konstruktor D')
15         super().__init__()
16 d = D()
```

Višestruko nasleđivanje i *super* nastavak

- Iz ispisa se vidi da će se pozvati konstruktor klase D, pa B, pa C i na kraju A
- Ovo se može zaključiti i pozivom atributa *__mro__*

```
1 print(D.__mro__)
2 #(<class '__main__.D'>,
3 #<class '__main__.B'>,
4 #<class '__main__.C'>,
5 #<class '__main__.A'>,
6 # <class 'object'>)
```

- Zbog mogućih problema, generalno pravilo je izbegavati višestruko nasleđivanje ako nema potrebe za njim.

Polimorfizam

- Polimorfizam se odnosi na sposobnost objekata da imaju različite oblike ili ponašanja u zavisnosti od konteksta u kojem se koriste
- U Pajtonu polimorfizam se postiže korišćenjem nasleđivanja i preklapanja (nadjačavanja) metoda
- Kroz polimorfizam se ”ne gleda” koji tip objekta je u pitanju, već samo da li ima metodu koji se poziva.

Polimorfizam

```
1  class Zivotinja:
2      def zvuk(self):
3          print('Zivotinja proizvodi zvuk.')
4  class Macka(Zivotinja):
5      def zvuk(self):
6          print('Mjau!')
7  class Pas(Zivotinja):
8      def zvuk(self):
9          print('Vau!')
10
11 macka = Macka()
12 pas = Pas()
13 zivotinja = Zivotinja()
14 for i in (macka, pas, zivotinja):
15     i.zvuk()
```

Hvala na pažnji!