Objektno programiranje
⟨ Nasljeđivanje klasa, moduli i paketi, rad s datotekama ⟩

Tihomir Žilić, Mario Essert, Vladimir Milić

Sveučilište u Zagrebu
Fakultet strojarstva i brodogradnje

Zagreb, 2020./2021.

Sadržaj

Nasljeđivanje klasa

```
class Potomak(Roditelj1, Roditelj2, ...):
    tijelo klase
```

- Roditeljska klasa (bazna klasa ili superklasa).
- Potomak klasa (podklasa), nasljeđuje strukture roditeljskih klasa.
- Potomak klasa može se dopuniti svojim elementima.

Primjer nasljeđivanja klasa

Klase:

```
>>> class C1: a=1    # roditeljska klasa
>>> class C2: b=2    # roditeljska klasa
>>> class C(C1,C2): # potomak klasa
        g=3
```

Objekti:

```
>>> c1=C1()     # objekt iz C1
>>> c2=C2()     # objekt iz C2
>>> c=C()       # objekt iz C
>>> c1.a        # objekt c1 ima a
1
>>> c2.b        # objekt c2 ima b
2
>>> print(c.a,c.b,c.g)  # objekt c ima a,b,g
1 2 3
```

Primjer nasljeđivanja klasa s istim imenima atributa

Klase:

```
>>> class C1: a=1    # roditeljska klasa
>>> class C2: b=2    # roditeljska klasa
>>> class C(C1,C2): # potomak klasa
        a=12
        g=3
```

Objekti:

```
>>> c1=C1()     # objekt iz C1
>>> c2=C2()     # objekt iz C2
>>> c=C()       # objekt iz C
>>> c1.a        # objekt c1 ima a
1
>>> c2.b        # objekt c2 ima b
2
>>> print(c.a,c.b,c.g)  # objekt c ima a,b,g
12 2 3
```

Prioriteti kod nasljeđivanja klasa

Istoimeni atributi prvog roditelja imaju prednost.

```
>>> class C1: a=1
>>> class C2: a=2
>>> class C(C1,C2):
        pass
>>> c=C()
>>> c.a
1
```

Podaci i metode unutar naslijeđenih klasa

Definiranje klasa:

```
class C1: a="Idem "    # roditeljska klasa 1
class C2: b="kući "    # roditeljska klasa 2
class C3: a="i "       # roditeljska klasa 3
class C(C1,C2,C3): # potomak klasa roditelja 1,2
    g="po "
    def metoda(self):
        self.z="bicikl "
        self.g="auto "
        ispis=C.a+super().b+C.g+self.g+C3.a+self.z
        print(ispis)          # ispis pri pozivu metode
    print(C1.a + C2.b + g) #ispis pri stvaranju klase
```

Poziv:

```
'Idem kući po'
>>> c=C()
>>> c.metoda()
'Idem kući po auto i bicikl'
```

Nasljeđivanje klasa s (_ , __) podacima

```
class Crtice:
    def __init__(self):
        self.no = 0
        self._one = 1
        self.__two = 2
>>> c=Crtice()
>>> print(dir(c)) # ispis identifikatora objekta 'c'
['_Crtice__two', '_one', 'no', ...]

class NasljedjeneCrtice(Crtice):
    def __init__(self):
        super().__init__() # ili Crtice.__init__(self)
        self.no = 'a'
        self._one = 'b'
        self.__two = 'abc'
>>> n=NasljedjeneCrtice()
>>> print(dir(n))  # ispis identifikatora objekta 'n'
['_Crtice__two', '_NasljedjeneCrtice__two', '_one', \
 'no', ...]
```

Primjer nasljeđivanja klasa s (__) istoimenim podacima

Klase:

```
>>> class C1: __a=1
>>> class C2: b=2
>>> class C(C1,C2):
        __a=12
        g=3
        print(C1._C1__a+C2.b+g+__a)
18
```

Objekti:

```
>>> c1=C1()    # objekt iz C1
>>> c2=C2()    # objekt iz C2
>>> c=C()      # objekt iz C
>>> c1._C1__a # objekt c1 ima __a
1
>>> c2.b       # objekt c2 ima b
2
>>> print(c._C1__a, c._C__a, c.b, c.g)
1 12 2 3
```

Primjer nasljeđivanja klasa s (__) metodama

Klase:

```
class A:
    def __test(self):
        print("test metoda iz klase A")
    def test(self):
        self.__test()
class B(A):
    def __test(self):
        print("test metoda iz klase B")
```

Objekti:

```
>>> b = B()          # stvaranje objekta
>>> b._A__test()     # test metoda iz klase A
>>> b._B__test()     # test metoda iz klase B
>>> b.test()         # test metoda iz klase A
```

Nasljeđivanja klasa s @classmethod dekoratorom

Unutar nasljeđenih klasa će se metode definirane @classmethod dekoratorom odnositi na klasu iz koje se metoda poziva.
Primjer:

```
class Times:                    # klasa
    factor = 1
    @classmethod
    def mul(cls,x):
        return cls.factor*x
class TwoTimes(Times):   # naslijeđena klasa
    factor = 2

x = Times.mul(4)        # poziv iz klase cls -> Times
y = TwoTimes.mul(4)     # poziv iz klase cls -> TwoTimes,
                        # tj. Times.mul(TwoTimes, 4) -> 8

print(x)
4
print(y)
8
```

Provjera porijekla objekata i klasa

Objekti: **isinstance**(objekt, (klase))

```
>>> isinstance(c1,C1)
True
>>> isinstance(c1,(C,C2))
False
```

Klase: **issubclass**(klasa, (roditeljske klase..))

```
>>> issubclass(C,C1)
True
>>> issubclass(C1,(C,C2))
False
```

Moduli i paketi

- **Modul**: datoteka s ekstenzijom **.py**, (npr. `ime_modula.py`), koja sadrži skup varijabli, funkcija, klasa i drugih objekata. Učitavanje modula **unutar druge .py datoteke** ostvaruje se:
  - **`import`** `ime_modula`
    `ime_modula` $\rightarrow$ {*math, sys, collections, copy, random, ...*}
    ```
    >>> import math
    >>> a=math.sin(3.7)
    ```
  - **`import`** `ime_modula` **`as`** `neko_ime`
    ```
    >>> import math as mt
    >>> a=mt.sin(1.2*mt.pi)
    ```
  - **`from`** `ime_modula` **`import`** `objekti`
    `objekti` (npr. *math*) $\rightarrow$ {*sin, tan, log, pi ..*}
    ```
    >>> from math import sin, tan, pi
    >>> a=sin(1.2*pi)
    ```
- **Paket**: skup modula pod istim imenom.

Moduli – primjer

Definiranje modula **ime_modula.py**:

```
# modul ime_modula.py za primjer

a=37                          # Varijabla
def A():                      # Funkcija
    print("Ja sam u A")
class K:                      # Klasa
    def B(self):
    print("Ja sam u K.B")
b=K()                         # Tvorba objekta
```

Pozivanje modula **ime_modula.py** unutar **poziv_modula.py**:

```
#poziv modula ime_modula.py

from ime_modula import a, A
k=99
print("Zbroj varijabli je: {}".format(k+a))
A()
```

```
>>>  Zbroj varijabli je: 136
Ja sam u A
```

Definiranje skripte istodobno kao programa i modula

Koristeći `__name__=='__main__'` pri dnu skripte može se svaka skripta definirati istodobno za pokretanje kao samostalni program i kao modul za učitavanje.

Definiranje skripte **`.py`**

```
definicija glavne funkcije
    ... poziv ostalih definicija ...

ostale definicije
    ... klase, funkcije, podatci ...

if __name__ == '__main__':
    ... izvrsi glavnu funkciju kao program
else:
    ... ucitaj ostale definicije kao modul
```

Primjer izvedbe skripte kao programa i modula

Definiranje skripte **modulime.py**

```
def glavni():
    print('Ovaj program je izvrsen samopokretanjem')
    print("__name__=",__name__)
def sporedni():
    print('Ovaj program je pokrenut kao vanjski modul ')
    print("__name__=",__name__)
if __name__ == '__main__':
    glavni()     # izvrsi funkciju glavni()
else:
    sporedni()   # izvrsi funkciju sporedni()
```

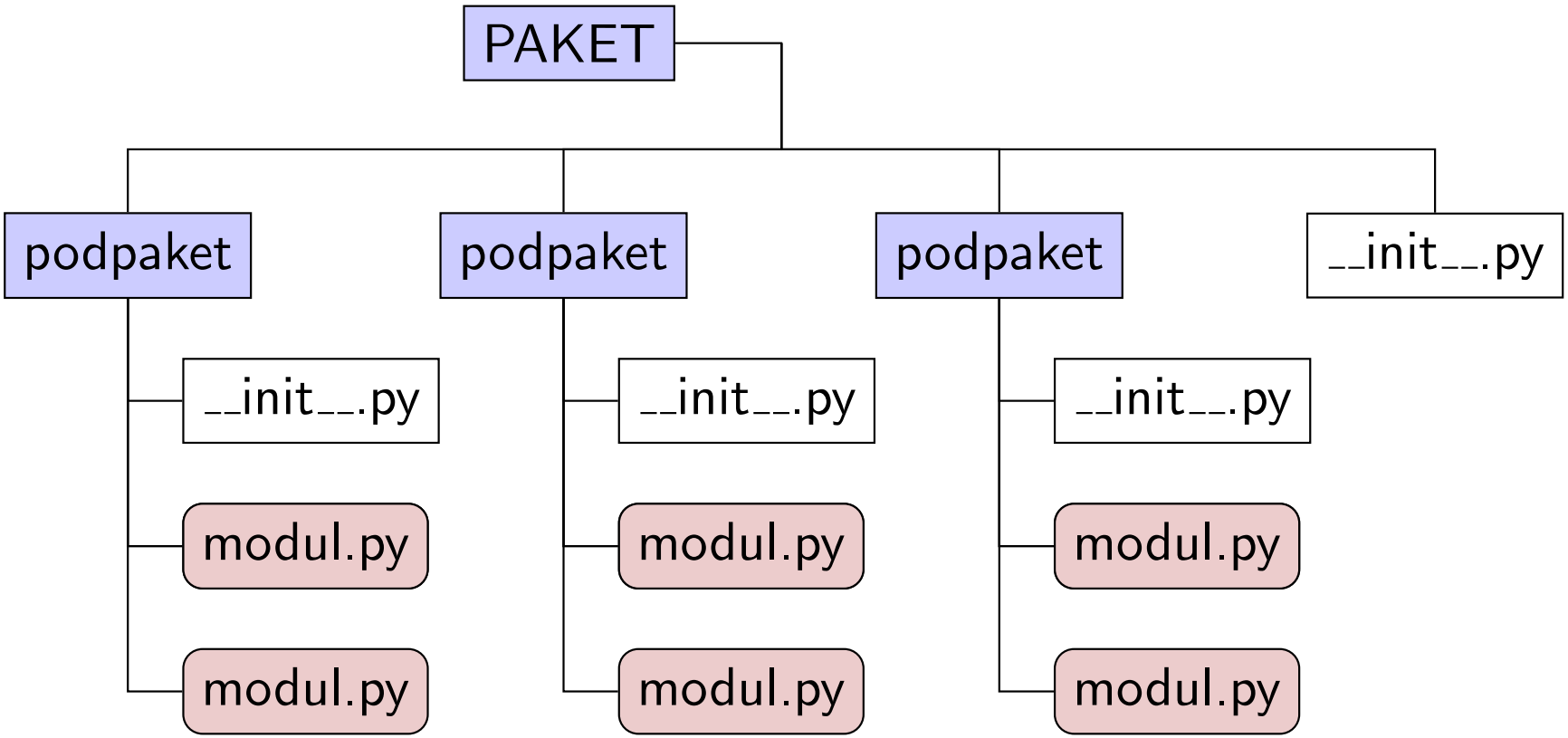
Direktno pokretanje skripte **modulime.py** iz interpretera *runfile('modulime.py')* ili terminala *python3 modulime.py*

```
>>> runfile('modulime.py')
Ovaj program je izvrsen samopokretanjem
__name__ = __main__
```

Pokretanje **modulime.py** kao modula

```
>>> import modulime
Ovaj program je pokrenut kao vanjski modul
__name__ = modulime
```

Paketi



Paketi i podpaketi su direktoriji i poddirektoriji na čvrstom disku računala, a inicijalizirani su obveznom __init__.py datotekom. U svakom paketu/podpaketu nalaze se razni moduli (.py datoteke). Učitavanje paketa/modula vrši se koristeći (dot) sintaksu, npr.

```
>>> import Paket.podpaket.modul
```

Rad s datotekama, tekstualne (txt,csv), binarne (zip,..)

1. Otvaranje datoteke, `f=open(ime_datoteke,mode)`
   dok mode može biti:
   - 'r' – čitanje,
   - 'w' – upisivanje teksta preko postojećeg,
   - 'a' – dodavanje teksta na kraj postojećeg,
   - 'b' – binarna datoteka; 't', tekstualna datoteka (engl. *default*)
   - '+' – čitanje/pisanje, 'r+' ili 'w+'.

   Na primjer otvaranje datoteke 'datoteka.txt' za čitanje:

   ```
   >>> f=open('datoteka.txt','r'),
   ```
2. čitanje datoteke `f.read()` ili upisivanje u nju `f.write(tekst)`
3. zatvaranje datoteke

   ```
   f.close()
   ```

Automatsko otvaranje i zatvaranje datoteke s naredbom with:

```
with open(ime_datoteke,mode) as f:
    ...citanje/upisivanje ...
```

Primjeri za rad s datotekama

Otvaranje, čitanje te zatvaranje datoteke:

```
>>> f=open('datoteka.txt','r')
    podaci = f.read()
    f.close()
```

Automatsko otvaranje, čitanje te zatvaranje datoteke s naredbom with:

```
>>> with open('datoteka.txt','r') as f:
        podaci = f.read()
```

Metoda `f.read()` čita cijelu datoteku u jedan string, dok `f.read(n)` čita n znakova iz datoteke u string, te svakim pozivom čita sljedećih n znakova (n je cijeli broj).
Metoda `f.readline()` čita jednu liniju datoteke.

Primjeri za rad s datotekama

Otvaranje, pisanje te zatvaranje datoteke:

```
>>> f= open("datoteka.txt",'w')
>>> f.write("Danas je lijep dan \n a i sutra će!")
>>> f.close()
```

Tekst se može upisivati u otvorenu datoteku ponavljanjem metode `f.write(tekst)`

Primjeri za rad s posebnim slovima

Uključivanjem paketa codecs.
Otvaranje, pisanje te zatvaranje datoteke:

```
>>> import codecs
>>> f= codecs.open("datoteka.txt",'w',"utf-8-sig")
>>> f.write("Danas je lijep dan \n a i sutra će!")
>>> f.close()
```

Automatsko otvaranje, čitanje te zatvaranje datoteke s naredbom with:

```
>>> with open('datoteka.txt','r') as f:
        podaci = f.read()
```

Rad s datotekama s puno numeričkih podataka

Iz paketa **numpy** pozove se metoda **loadtext()**.
Tip podatka - polje (engl. *array*).

Primjer:

```
>>> import numpy as np
>>> data=np.loadtxt(file_name, dtype=float)
>>> podaci=data[:,0] #ucitavanje prvog \
                     #stupca iz datoteke
    ...
```