

# Algoritmi i strukture podataka

Objektno orijentisano programiranje

Katedra za informatiku, Fakultet tehničkih nauka, Novi Sad

2023

# Ciljevi

- Robusnost
- Adaptivnost
- Ponovna iskoristivost

# Principi objektno-orijentisanog dizajna

- Modularnost
- Apstrakcija
- Enkapsulacija

# Modularnost


- Podela sistema u nezavisne funkcionalne celine
- Benefiti:
  - Olakšava testiranje komponenti
  - Omogućava ponovnu iskoristivost
  - Smanjuje dupliciranje koda
- Python modul:
  - Skup srodnih funkcija i klasa
  - Definiše se unutar jednog py fajla
  - Upotreba:

```
import re from os
import * from math import pi, sqrt
```

# Apstrakcija


- Izdvajanje najvažnijih karakteristika sistema
- Kreira apstraktne tipove podatka (ATP)
- ATP specifikuju **šta** operacije rade, ali ne i **kako**
- Skup operacija koje ATP definiše naziva se **interfejs**
- Python:
  - Duck typing
  - Apstraktne bazne klase

# Interfejs- slikovito objašnjenje




Consider the following situation:

162



*You are in the middle of a large, empty room, when a zombie suddenly attacks you.*

*You have no weapon.*



*Luckily, a fellow living human is standing in the doorway of the room.*

*"Quick!" you shout at him. "Throw me something I can hit the zombie with!"*

Now consider:  
You didn't specify (nor do you care) exactly **what** your friend will choose to toss;  
...But it doesn't matter, as long as:

- It's something that **can** be tossed (He can't toss you the sofa)
- It's something that you can grab hold of (Let's hope he didn't toss a shuriken)
- It's something you can use to bash the zombie's brains out (That rules out pillows and such)


It doesn't matter whether you get a baseball bat or a hammer -  
as long as it implements your three conditions, you're good.

**To sum it up:**

When you write an interface, you're basically saying: "I need something that..."

[share](#) [improve this answer](#)

answered Jan 9 '13 at 19:14



**Yehuda Shapira**  
7,002 ● 4 ● 39 ● 61

Algoritmi i strukture podataka

6

# Enkapsulacija

- Detalji implementacije ostaju sakriveni
- Omogućava nezavisnost prilikom implementacije komponenti
- Python:
  - Na nivou preporuke
  - članovi klase čije ime počinje donjom crtom su privatni i nisu namenjeni upotrebi izvan klase (npr. `_item`)

# Terminologija

- Objekat je **instanca** klase
- Metode klase definišu **ponašanje**
- Atributi klase definišu **stanje**



# Identifikator *self*

- Svaka klasa može imati više instanci
- (Isto značenje kao **this** u Javi)
- ***self*** predstavlja trenutnu instancu klase
- Za razliku od Jave, upotreba identifikatora **self** je obavezna prilikom pozivanja metoda i pristupa atributima

# Nasleđivanje

- Omogućava modularnu i hijerarhijsku organizaciju
- Nova klasa definiše se na osnovu postojeće
- Polazna klasa se naziva **bazna**, **predak** ili **superklasa**
- Izvedena klasa naziva se **potklasa** ili **klasa potomak**

# Nasleđivanje

- Opšti format:

```
class NazivKlase(A, B, ...):  
    # telo klase ...
```

- new-style vs old-style

```
class A:  
    # old-style
```

- 

```
class B(object):  
    # new-style
```

- Pristup roditeljskoj klasi pomoću **super**

# Konstruktori

- Kreiraju instance klase
- Poziv se svodi na poziv metode **`__init__`**
- Primer:

```
objekat = Klasa()
```

# Primer klase - get i set metode

```
class Student(object):
    def __init__(self, firstname, lastname):
        self._firstname = firstname
        self._lastname = lastname

    def get_firstname(self):
        return self._firstname

    def set_firstname(self, firstname):
        self._firstname = firstname

    def get_lastname(self):
        return self._lastname

    def set_lastname(self, lastname):
        self._lastname = lastname

    def introduce_yourself(self):
        return 'Ja sam ' + self._firstname + ' ' + self._lastname
```

```
If __name__=='__main__':
    student = Student('Minja', 'Minjic')
    student.set_firstname('Steva')
    print(student.get_lastname())
    student.introduce_yourself()
```

# Primer klase - properties

```
class Student(object):
    def __init__(self, firstname, lastname):
        self._firstname = firstname
        self._lastname = lastname

    @property
    def firstname(self):
        return self._firstname
    @firstname.setter
    def firstname(self, firstname):
        self._firstname = firstname

    @property
    def lastname(self):
        return self._lastname
    @ lastname.setter
    def lastname(self, lastname):
        self._lastname = lastname

    def introduce_yourself(self):
        return 'Ja sam ' + self._firstname + ' ' + self._lastname
```

```
If __name__ == '__main__':
    student = Student('Minja', 'Minjic')
    student.firstname = 'Steva'
    print(student.lastname)
    student.introduce_yourself()
```

# Preklapanje operatora

- Omogućava izmenu semantike postojećih operatora
- Postiže se redefinisanjem specijalnih metoda
- Primer:
  - Operator + označava sabiranje među brojevima (pišemo  $2+3$ )
  - Primećujemo da operator + nema značenje među stringovima (sabiranje stringova nije moguće)
  - Za stringove je definisana operacija konkatencije (spajanja)
  - Mogli bismo tu operaciju da implementiramo kao metodu
  - `"abc".concatenate("efg")`
  - Ili da redefinišemo značenje za operator + u skupu stringova pa da možemo pisati `"abc" + "efg"`

# Preklapanje operatora

- Dakle, operator `+` u zavisnosti od tipova na koje se primenjuje dobija različita značenja:
  - Sabiranje u skupu brojeva  $3 + 11 = 14$
  - Konkatencija stringova `"abc" + "efg" = "abcdefg"`
  - Spajanje listi `[1, 2, 3] + [4] = [1, 2, 3, 4]`
  - Možemo redefinisati značenje operatora `+` i za proizvoljan kontekst naše klase



# Iteratori

- Ponoviti generator funkcije sa Osnova programiranja
- Omogućavaju pristup elementima kolekcije
- Redefiniše se metoda **`__next__`**
- Metoda vraća sledeći element u kolekciji ili izaziva **StopIteration** izuzetak
- Druga mogućnost je da se implementira **`__iter__`** metoda.

# Primer

- Napisati klasu **ComplexNumber** koja predstavlja kompleksan broj.
- Klasa treba da sadrži:
  - metode za pristup realnom i imaginarnom delu
  - metodu **`__str__`**
  - podršku za osnovne računske operacije – sabiranje, oduzimanje, množenje i deljenje

# Zadatak 1

- Napisati klasu **Rectangle** koja reprezentuje pravougaonik.
- Klasa sadrži metode za izračunavanje obima i površine pravougaonika.
- Na osnovu napisane klase izvesti klasu **Square**.

## Zadatak 2

- Proširiti klasu **Student** iz primera da podrži evidentiranje ocena i ispis proseka.
- Evidentiranje ocena se ostvaruje kroz metode upisi\_ocenu (kojoj se prosleđuje nova ocena studenta) i ponisti\_ocenu (kojoj se prosleđuje ocena koja se poništava).
- Na osnovu ocena studenta, omogućiti ispis, računanje i korekciju proseka (usled brisanja ili upisivanja ocene).

# Zadatak 3

\*zadatak preuzet sa predmeta Uvod u medicinsku informatiku

- Napisati klasu ***Player*** koja ima:
  - attribute:
    - **health** (*float*)
    - **mana** (*float*)
  - metode:
    - konstruktor, koji inicijalizuje attribute **health** i **mana** na vrednost prosleđenih parametara
    - `__str__` metodu, koja formatirano ispisuje vrednost atributa **health** i **mana**
- Napisati klasu ***Item*** koja ima:
  - attribute:
    - **value** (*float*)
  - metode:
    - konstruktor, koji inicijalizuje atribut **value** na vrednost prosleđenog parametra
    - `__str__` metodu, koja formatirano ispisuje vrednost atributa **value**
    - metodu `use(self, player)`, bez implementacije

# Zadatak 3

\*zadatak preuzet sa predmeta Uvod u medicinsku informatiku

- Napisati klasu **Food** koja nasleđuje klasu **Item**, a koja ima:
  - metode:
    - redefinisani metodu *use(self, player)*, koja uvećava **health** prosleđenog player-a za:  
 $health = health + value$
- Napisati klasu **Potion** koja nasleđuje klasu **Item**, a koja ima:
  - dodatne attribute:
    - **type** (*string*), koji može imati vrednost "health" ili "mana"
  - metode:
    - prošireni konstruktor koji dodatno inicijalizuje atribut **type** na vrednost prosleđenog parametra
    - proširenu `__str__` metodu, koja dodatno formatirano ispisuje vrednost atributa **type**
    - redefinisani metodu *use(self, player)*, koja menja attribute prosleđenog player-a na sledeći način:
      - a) ako **type** atribut potion-a ima vrednost "health" tada **health** atribut player-a treba uvećati za:  
 $health = health + value$
      - b) ako **type** atribut potion-a ima vrednost "mana" tada **mana** atribut player-a treba uvećati za:  
 $mana = mana + value$

# Zadatak 3

\*zadatak preuzet sa predmeta Uvod u medicinsku informatiku

- Za svaku od klasa, pristup atributima implementirati kroz @property dekorator (slajd 14).
- Test funkcija treba da se izvrši bez interakcije sa korisnikom u sledećim koracima:
  1. napraviti player-a sa početnim stanjem (health: 100, mana: 100)
  2. ispisati player-a
  3. Napraviti listu item-a:
    - food (value: 100.0)
    - potion (value: 200.0, type: "health")
    - potion (value: 300.0, type: "mana")
  4. for petljom proći kroz item-e na sledeći način:
    - ispisati item
    - pozvati metodu *use* nad player-om
    - ispisati player-a

```
health: 100.0, mana: 100.0

value: 100.0
health: 200.0, mana: 100.0

value: 200.0 type: health
health: 400.0, mana: 100.0

value: 300.0 type: mana
health: 400.0, mana: 400.0
```