Objektno-orijentisano programiranje i Python

Slajdovi za predmet Osnove programiranja

Katedra za informatiku, Fakultet tehničkih nauka, Novi Sad

2022.

Ciljevi

- savlađivanje osnovnih pojmova objektno-orijentisanog programiranja

Pojam objekta

- objekat je akter u realnom sistemu koji opisujemo programom
- na primer, naš program treba da rukuje dužima i krugovima u Dekartovom koordinatnom sistemu
- šta je duž? šta je krug?
  - matematička definicija: skup tačaka koje...
  - programerska definicija: koji podaci su nam dovoljni da opišu duž ili krug?

Pojam objekta 2

- duž je određena pomoću dve krajnje tačke
- krug je određen centrom (tačka) i poluprečnikom
- tačka je određena Dekartovim koordinatama
- $\Rightarrow$ tačka je objekat opisan svojim koordinatama
- program može da rukuje sa više tačaka istovremeno
- **svaka tačka** biće poseban objekat
- šta je zajedničko svim tačkama?
  - struktura (podaci kojima su opisane)
  - ponašanje (operacije nad njima)

Pojam klase

- strukturu i ponašanje svih tačaka opisaćemo na jednom mestu
- taj opis nazvaćemo klasa
- klasa predstavlja novi tip podataka
- konkretni **objekti** su instance (primerci) **klase**

Pojam klase 2

```
class Point: # klasa Point opisuje sve tacke u programu
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def equals(self, other):
        return self.x == other.x and self.y == other.y

a = Point(5, 3)  # a je objekat klase Point
b = Point(3, 4)  # b je objekat klase Point
c = Point(5, 3)  # c je objekat klase Point
print(a.equals(b))
print(a.equals(c))
```

Pojam klase

- atributi klase: podaci koji opisuju stanje objekta
  - atributi klase `Point`: `x`, `y`
- metode klase: funkcije koje opisuju ponasanje objekta
  - metode klase `Point`: `equals`

Šta je self

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def equals(self, other):
        return self.x == other.x and self.y == other.y
```

- self označava onaj objekat čija metoda je pozvana
  - self = "ja"
- self.x: vrednost mog atributa x
- other.x: vrednost atributa x objekta other

Šta je self 2

- ko je self a ko je other?

```
print(a.equals(b))
```

- poziva se metoda equals objekta a
- parametar poziva metode je objekat b
- a je self
- b je other

Šta je self [3]

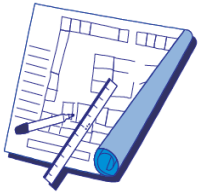
- prvi parametar svake funkcije koja je deo klase je self

```
class Point:
    ...
    def equals(self, other):
        return self.x == other.x and self.y == other.y
```

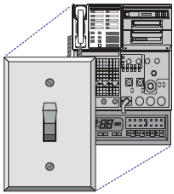
Principi objektno-orijentisanog dizajna

- modularnost
- apstrakcija
- enkapsulacija

Modularity



Abstraction



Encapsulation

Principi objektno-orijentisanog dizajna $_2$

- klasa sadrži atribute i metode
- $\Rightarrow$ podaci i operacije nad podacima se nalaze na jednom mestu
- pristup podacima je kontrolisan; mogu se menjati samo putem postojećih operacija

Šta je __init__

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

- `__init__` je metoda koja se automatski poziva kada se objekat kreira u memoriji
- konstruktor objekta

Primer: tačka, duž, krug

```
import math

class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def equals(self, other):
        return self.x == other.x and self.y == other.y

    def distance(self, other):
        return math.sqrt((self.x-other.x)**2 +
            (self.y-other.y)**2)
```

Primer: tačka, duž, krug $_2$

```
class Circle:
    def __init__(self, center, radius):
        self.center = center
        self.radius = radius

    def contains(self, point):
        return center.distance(point) <= radius
```

Primer: tačka, duž, krug $_3$

```
class LineSegment:
    def __init__(self, point1, point2):
        self.point1 = point1
        self.point2 = point2

    def contains(self, point):
        # proveri da li je tacka na duzi
```

Nasleđivanje

- šta je zajedničko za tačku, krug i duž?
- svi su geometrijski pojmovi
- mogu se nacrtati na ekranu
- algoritam za crtanje nije isti u sva tri slučaja

Nasleđivanje

- zajedničke osobine više klasa možemo grupisati u posebnu roditeljsku klasu
- postojeće klase nasleđuju roditeljsku klasu
- nasleđuju atribute i metode
- i mogu da dodaju specifično ponašanje

Primer: tačka, duž, krug 4

```
class Shape:
    def __init__(self):
        # zajednicki atribut
        self.color = '000000'

    def draw(self):
        # apstraktni oblik ne ume da se nacrta
        raise NotImplementedError(
            'Naslednik mora ovo da redefinise!')
```

Primer: tačka, duž, krug $_5$

```
class Point(Shape):
    ...
    def draw(self):
        # crtanje tacke

class Circle(Shape):
    ...
    def draw(self):
        # crtanje kruznice

class LineSegment(Shape):
    ...
    def draw(self):
        # crtanje duzi
```

Primer: tačka, duž, krug $_6$

```
class Point(Shape):
    def __init__(self):
        # poziv roditeljskog konstruktora
        super(Point, self).__init__()

    def draw(self):
        # crtanje tacke
```

Polimorfizam

```
# predstavlja crtez koji se sastoji iz liste figura
class Drawing:
    def __init__(self):
        # elementi liste bice naslednici Shape-a
        self.shapes = []

    def drawEverything(self):
        # prodji kroz listu Shape-ova
        for shape in self.shapes:
            # on ce sam znati kako da se nacrta
            shape.draw()
```

Polimorfizam 2

```
drawing = Drawing()
p1 = Point(3, 4)
p2 = Point(6, 7)
p3 = Point(2, 2)
ls1 = LineSegment(p1, p2)
c1 = Circle(p2, 4)
drawing.shapes.append(p3)
drawing.shapes.append(ls1)
drawing.shapes.append(c1)

drawing.drawEverything()
```

- bez polimorfizma petlja za iscrtavanje figura bi morala da sadrži ogroman if/elif/elif/...

Strukturno vs. objektno-orijetisano programiranje

- osnovni koncepti **strukturnog programiranja**:
  - sekvenca (blok naredbi)
  - selekcija (grananje)
  - iteracija (petlje)
- dodajmo tu još i
  - modularizaciju (dekompoziciju na potprograme)

- **objektno-orijentisano programiranje** se oslanja na sve to i dodaje
  - klase i objekte
  - nasleđivanje
  - enkapsulaciju
  - apstrakciju

Primer: studentska služba

- implementiramo referentni projekat pomoću OOP

```
class Student:
    def __init__(self):
        self.indeks = ''
        self.ime = ''
        self.prezime = ''
        self.roditelj = ''
        self.jmbg = ''
        self.adresa = ''
        self.telefon = ''
        self.email = ''
        self.godina = 0

    def upisi_narednu_godinu(self):
        self.godina += 1
```

Primer: studentska služba 2

```
class Student:
    ...
    def to_string(self):
        return "|".join(s.indeks, s.ime, s.prezime,
            s.roditelj, s.jmbg, s.adresa, s.telefon,
            s.email, s.godina')

    def from_string(line):
        # Ovo je statička metoda: nije potreban objekat
        # da bi se pozvala
        s = Student()
        s.indeks, s.ime, s.prezime, s.roditelj, s.jmbg,
            s.adresa, s.telefon, s.email, s.godina = \
            line.split("|")
        return s
```

Primer: studentska služba 3

```
# učitavanje studenata iz fajla
studenti = []
for line in open('studenti.txt', 'r'):
    s = Student.from_string(line[:-1])
    studenti.append(s)
```

- gde staviti operacije nad listom studenata?
- $\Rightarrow$ klasa `StudentInformationService`

Primer: studentska služba 4

```
class StudentInformationService:
    def __init__(self):
        self.studenti = []

    def load(filename):
        studenti = []
        with open(filename, 'r') as student_file:
            for line in student_file.readlines():
                s = Student.from_string(line[:-1])
                self.studenti.append(s)

    def upis_naredne_godine():
        for s in self.studenti:
            if s.ima_uslov():
                s.upisi_narednu_godinu()
```