Objektno-orijentisano programiranje u Python-u

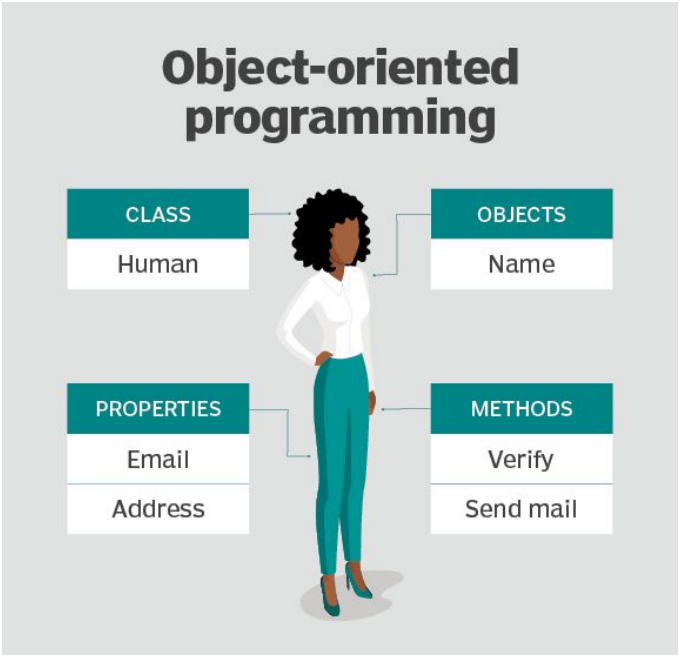
IRM - Srednja grupa 2019/2020
Filip Popović 89/2016

Uvod u OOP

Šta je OOP?

Do sada smo navikli da zadajemo redom komande koje će interpreter izvršavati jednu za drugom. Ovakva ideja programiranja naziva se **paradigma proceduralnog programiranja**. Međutim, ona nije jedina programerska paradigma.

Paradigma objektno-orijentisanog programiranja zasnovana je na **entitetima** i na **interakcijama** između tih entiteta. Ovim entitetima se zadaju **stanja** i **ponašanja** i osposobljuju se za interakciju. Na programeru je zadatak da implementira ova stanja i ponašanja, a zatim pomoću njih da formira neki sistem.



Ideje i ciljevi OOP-a

Paradigma OOP-a zasnovana na entitetima koji se nazivaju **objekti**. Ovi objekti imaju odgovarajuća stanja (ime učenika, rasa psa, brojno stanje novčanica u novčaniku, itd.) kao i odgovarajuća ponašanja (odazivanje, kretanje, ubacivanje novčanice u novčanik, itd.).

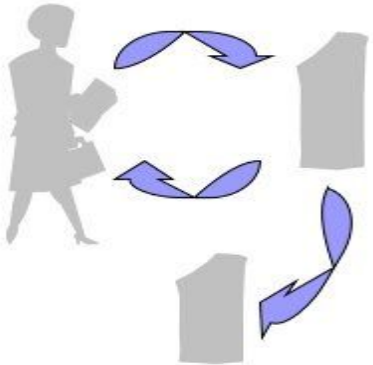
Objekti se konstruišu po šablonu zadatom u **klasama**. Klasa predstavlja skicu po kojoj se formira neki objekat.

Cilj OOP-a jeste da omogući lakše projektovanje **problema iz života** pomoću koda. Za neke probleme iz stvarnog sveta, gotovo je nemoguće uraditi pomoću proceduralnog programiranja.

Dodatno, formiranje klasa zna drastično da poboljša uređenost i čitljivost koda.

Modularnost je još jedna ključna stavka koja se postiže korišćenjem OOP-a. Ona predstavlja mogućnost nezavisnog razvoja pojedinih delova koda. Bilo kakve neregularnosti u okviru jednog modula, neće uticati na celokupni konačni program.

- Procedural



Withdraw, deposit, transfer

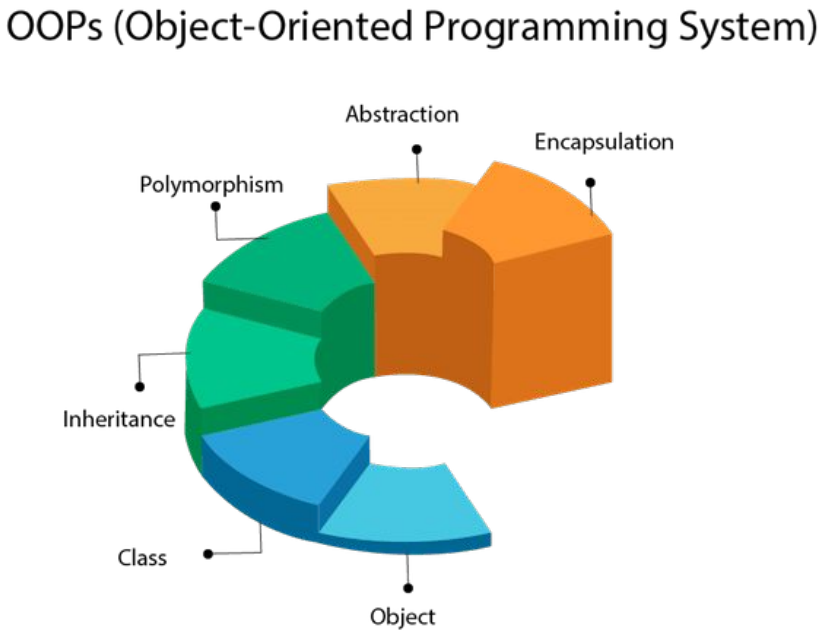
- Object Oriented

Customer, money, account

Principi OOP-a

Pored klasa i objekata, ključnu ulogu u razvoju programa u ovoj paradigmi imaju **principi OOP**-a.

1. Nasleđivanje
2. Polimorfizam
3. Apstrakcija
4. Enkapsulacija



Klase i Objekti

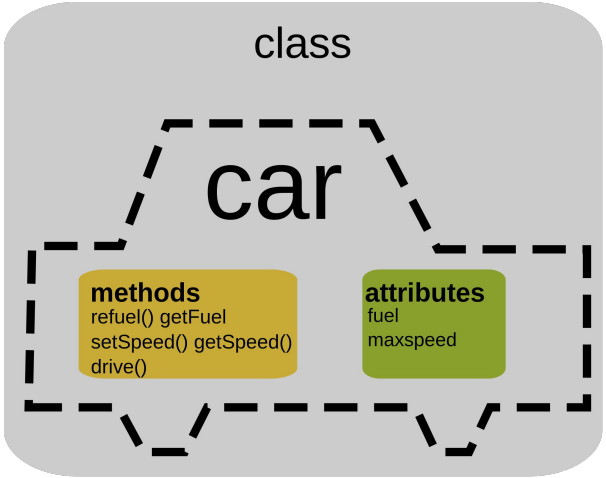
Klase

Klasa je tip koji predstavlja skicu ili šablon po kome se prave neki objekti.

U definiciji klase, nalazi se sam kod kojim se definišu stanja (atributi) i ponašanja(metode).

Ključna reč koja se koristi jeste **class**.

Za primer neke klase možemo uzeti ideju o automobilu. Znamo da svaki automobil mora imati točkove i sedišta kao i da ima mogućnost da se dopuni gorivom i da se vozi.



Primer 1.

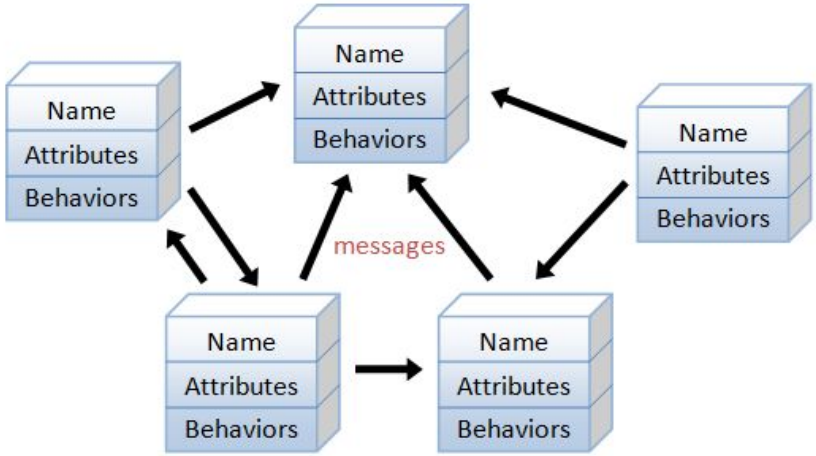
Napisati program u kome se definiše klasa **Automobil.**

```
1    class Automobil:
2        pass
3
4
```

Objekti

Objekti su sami “oživljeni” entiteti koji predstavljaju instancu klase. Dakle, sami objekti se konstruišu po specifikacijama u klasi, te poseduju sva stanja i ponašanja zadata u odgovarajućoj klasi.

Sami objekti mogu da se “imenuju” tako što ih vežemo za neku promenljivu. Tada kažemo da se ta promenljiva naziva **referenca** na objekat.



An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages


Class
Definition of objects that share structure, properties and behaviours.
Building
class
Dog
class
Computer
class
Instance
Concrete object, created from a certain class.
Empire State
instance of Building
Lassie
instance of Dog
Your computer
instance of Computer

Konstruktori

Definisanje klase nije potreban i dovoljan uslov za kreiranje objekta. Kako bi se kreirao objekat, neophodno je naglasiti interpreteru da želimo da imamo instancu neke klase. To radimo pomoću konstruktora.

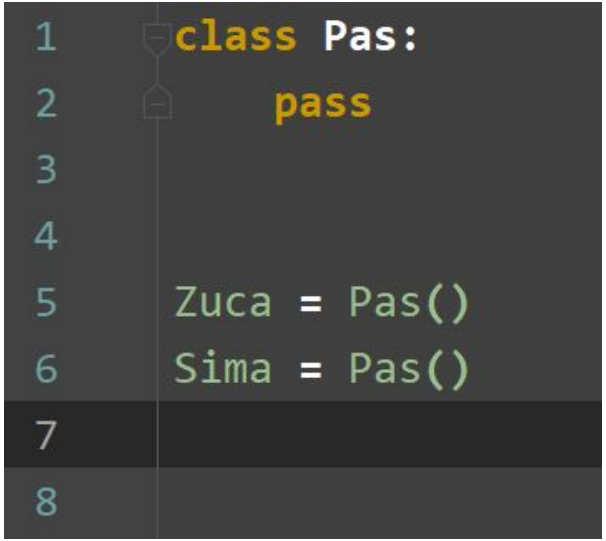
Konstruktori su specijalizovane metode (ponašanja/funkcije) čiji je glavni cilj da naprave objekat po specifikacijama zadatih u klasi.

Za svaku klasu, iako nije eksplicitno navedeno, postoji podrazumevani konstruktor koji je neparametrizovan, odnosno ne prihvata nikakve argumente.

Kako bi se pozvao podrazumevani konstruktor neophodno je navesti naziv klase čiji objekat želimo napraviti, a zatim dodati otvorenu i zatvorenu zagradu.

Primer 2.

Napisati program u kome se definiše klasa **Pas**. U glavnom delu programa korišćenjem podrazumevanog konstruktora napraviti dva imenovana objekta tipa **Pas**, ***Zuca*** i ***Sima***.



Parametrizovani konstruktori

Definicija konstruktora realizuje se u metodi pod nazivom **__init__(self)**. Ovo je ugrađena metoda koja postoji kao predefinisana za svaki tip i služi za konstruisanje objekata.

Kako je članica neke klase, sama metoda mora imati bar parametar **self**.

Ukoliko nam je potrebno, podrazumevanu __init__() metodu možemo izmeniti u skladu sa situacijom. Neretko se dešava da želimo da prosledimo neke dodatne parametre konstruktoru kako bismo bolje precizirali neki objekat.

Iznad navedeno postižemo dodavajući drugih argumenata posle navođenja ključne reči self u zagradama.

Primer 3.

Napisati program u kome se definiše klasa **Ucenik**. Za svakog učenika poznato je **ime**, **prezime** i **odeljenje** (u formatu “razred/odeljenje”). U glavnom delu programa korišćenjem parametrizovanog konstruktora napraviti učenika ***Marko***.

Primer 3.

```
1   class Ucenik:
2       def __init__(self, ime, prezime, odeljenje):
3           self.ime = ime
4           self.prezime = prezime
5           self.odeljenje = odeljenje
6
7
8   Marko = Ucenik("Marko", "Matovic", "VI/2")
9
10
```

Referenca na trenutni objekat - self

U prethodnom primeru bilo je reči o ključnoj reči **self**. Ovo je parametar koji će se uvek navoditi u zaglavlju klasnih metoda (metoda koje definišu ponašanje objekata).

Self predstavlja referencu na trenutni objekat s kojim interpreter radi. Možemo zamisliti da je to promenljiva koja pokazuje na baš taj objekat u datom trenutku.