Programação Orientada a Objetos

Prof. Márcio Miguel Gomes

Endereços de Memória

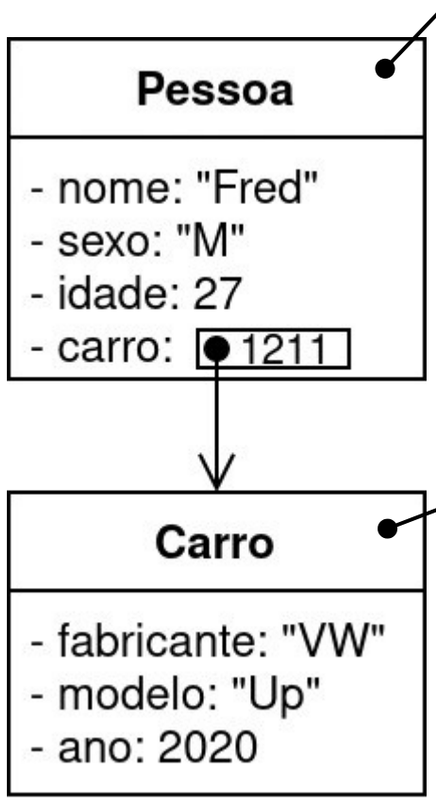
- Ao criarmos um programa, cada variável está associada a:
  - um nome
  - um valor
  - um tipo
  - um endereço de memória

- Exemplos
  - i = 5
  - txt = "ABCD"
  - nr = 12.34

Endereço	Nome	Valor
1199	...	...
1200	i	5
1201		
1202	txt	'A'
1203		'B'
1204		'C'
1205		'D'
1206	nr	12.34
1207		
1208		
1209		
1210	...	...

Endereços de Memória

- Objetos seguem a mesma regra
- Exemplo
  - p = Pessoa('Fred', 'M', 27)
  - c = Carro('VW', 'Up', 2020)
  - p.set_carro(c)



Endereço	Nome	Valor
1199	...	...
1200	nome	'F'
1201		'r'
1202		'e'
1203		'd'
1204	sexo	'M'
1205	idade	27
1206		
1207	carro	1211
1208		
1209	...	...
1210	...	...
1211	fabricante	'V'
1212		'W'
1213	modelo	'U'
1214		'p'
1215	ano	2020
1216		
1217	...	...

Instanciação Dinâmica

- Ao **invocarmos** o **construtor** de uma classe, o gerenciador de memória do sistema operacional **aloca** uma área da **memória RAM** que será destinada a **armazenar** os **dados** do **objeto**, ou seja, o **valor** de seus **atributos**

- Esta operação é chamada de **instanciação dinâmica**, e **retorna** o **endereço de memória inicial do bloco alocado**

- Ao **armazenarmos** o endereço de memória em uma **variável**, temos uma **referência** para o objeto, e não uma **cópia** dele

- Esse **bloco** de **memória** fica **alocado** para uso do **objeto** enquanto ele **existir**

Instanciação Dinâmica

- A partir da classe Carro

```
class Carro:
  def __init__(self, fabricante, modelo, ano):
    self._fabricante = fabricante
    self._modelo = modelo
    self._ano = ano

  def get_fabricante(self):
    return self._fabricante

  def get_modelo(self):
    return self._modelo

  def get_ano(self):
    return self._ano

  def exibe_dados(self):
    print('Fabricante:', self._fabricante)
    print('Modelo:', self._modelo)
    print('Ano:', self._ano)
    print()
```

Instanciação Dinâmica

- Se utilizarmos sempre a **mesma variável** para armazenar referências a **novos objetos** criados **dinamicamente**, **perdemos** a referência para os **objetos anteriores**, embora eles **continuem** alocados na **memória**
- **Problema**: como acessar os dados do “VW Up” ou do “Fiat Mobi”?

```
from carro import Carro

c = Carro('VW', 'Up', 2020)
c = Carro('Fiat', 'Mobi', 2019)
c = Carro('Ford', 'Ka', 2016)

print('Modelo:', c.get_modelo())
```

Instanciação Dinâmica

- Para termos acesso a **todos os objetos** criados **dinamicamente**, precisamos **armazenar** suas **referências** em uma estrutura de **coleção** de dados, como ***arrays*** ou **listas**

```
from carro import Carro

carros = list()

carros.append(Carro('VW', 'Up', 2020))
carros.append(Carro('Fiat', 'Mobi', 2019))
carros.append(Carro('Ford', 'Ka', 2016))

for carro in carros:
  print('Modelo:', carro.get_modelo())
```

Instanciação Dinâmica

- A instanciação **dinâmica** é muito **útil** quando **não sabemos** a **quantidade** de objetos que devem ser **instanciados**. Normalmente ocorre em duas situações
- **Leitura** de dados do usuário até que **determinada situação ocorra**
- **Carregamento** de dados a partir de **arquivos** ou **banco de dados**

```
from carro import Carro

carros = list()
continuar = 's'

while continuar in 'Ss':
  fab = input('Informe o fabricante: ')
  mod = input('Informe o modelo: ')
  ano = int(input('Informe o ano: '))
  carros.append(Carro(fab, mod, ano))
  continuar = input('Continuar? S/N: ')
```

```
print('\n** Carro(s) informado(s) **')
for carro in carros:
  print('Modelo:', carro.get_modelo())
```

Localização de Objetos

- Visto que **não** temos mais uma **variável específica** para **cada** objeto instanciado, como podemos **localizar** um **objeto** específico?
- Podemos **percorrer** toda a coleção **procurando** um ou mais objetos que se enquadrem nos **critérios desejados**, consultando seus **atributos**
- Podemos utilizar o recurso de ***list comprehension***, gerando **dinamicamente** uma **nova lista** com **referências** para os **objetos desejados** com base nos mais **variados critérios**

Percorrendo uma lista

- Utilizando o comando **for/in** juntamente com **if**

```
from carro import Carro

carros = list()

carros.append(Carro('VW', 'Up', 2020))
carros.append(Carro('Fiat', 'Mobi', 2019))
carros.append(Carro('Ford', 'Ka', 2016))

modelo = input('Informe o modelo desejado: ')

for carro in carros:
  if carro.get_modelo() == modelo:
    carro.exibe_dados()
    break
else:
  print(f'\nModelo "{modelo}" não localizado!\n')
```

Percorrendo uma lista

- Utilizando ***list comprehension***

```
from carro import Carro

carros = list()

carros.append(Carro('VW', 'Up', 2020))
carros.append(Carro('Fiat', 'Mobi', 2019))
carros.append(Carro('Ford', 'Ka', 2016))

modelo = input('Informe o modelo desejado: ')

sel = [c for c in carros if c.get_modelo() == modelo]
if len([c.exibe_dados() for c in sel]) == 0:
  print(f'\nModelo "{modelo}" não localizado!\n')
```

Atividade

A classe "Veículo" possui os atributos tipo (M = moto, C = carro, V = van), modelo, cor e ano_fabricacao, todos eles passados como parâmetros do construtor. O método idade() calcula e retorna a idade do veículo em anos, pela expressão idade = ano_atual - ano_fabricacao. Em uma estrutura de repetição, instancie dinamicamente veículos com dados informados pelo usuário e os armazene em uma lista. Ao final de cada iteração, pergunte ao usuário se ele deseja continuar adicionando veículos, e encerre a entrada de dados caso a resposta seja "N". Em seguida:

a) Mostre apenas o modelo dos veículos de cor azul

b) Calcule e mostre a idade média dos veículos

c) Mostre todos os dados dos veículos de um tipo específico informado pelo usuário

d) Mostre o modelo e idade dos veículos com 10 anos ou mais

e) Mostre a quantidade de veículos por tipo

f) Mostre todos os dados do veículo mais novo

g) Mostre tipo e modelo dos veículos brancos com idade entre 3 e 5 anos