Programação Orientada a Objetos

Prof. Márcio Miguel Gomes

Associação entre Classes

- Até o momento, nossas classes possuíam **atributos** de tipos de **dados primitivos**, como textos, números inteiros e números fracionários
- Porém, em cenários mais complexos, as classes podem ter **atributos** que são representados por **outras classes**, que por sua vez, possuem **atributos e métodos próprios** e **independentes**
- Nesse caso, costumamos dizer que a **classe principal** possui um relacionamento "**tem um**" com **outra classe**

Associação entre Classes

- Se uma **pessoa tem um carro**, isso indica que a classe **Pessoa tem uma associação** com a classe **Carro**
- Esse tipo de **relacionamento** entre classes é importante porque define como as classes **interagem** entre elas nas aplicações, mantendo a **integridade** e **independência** de cada objeto

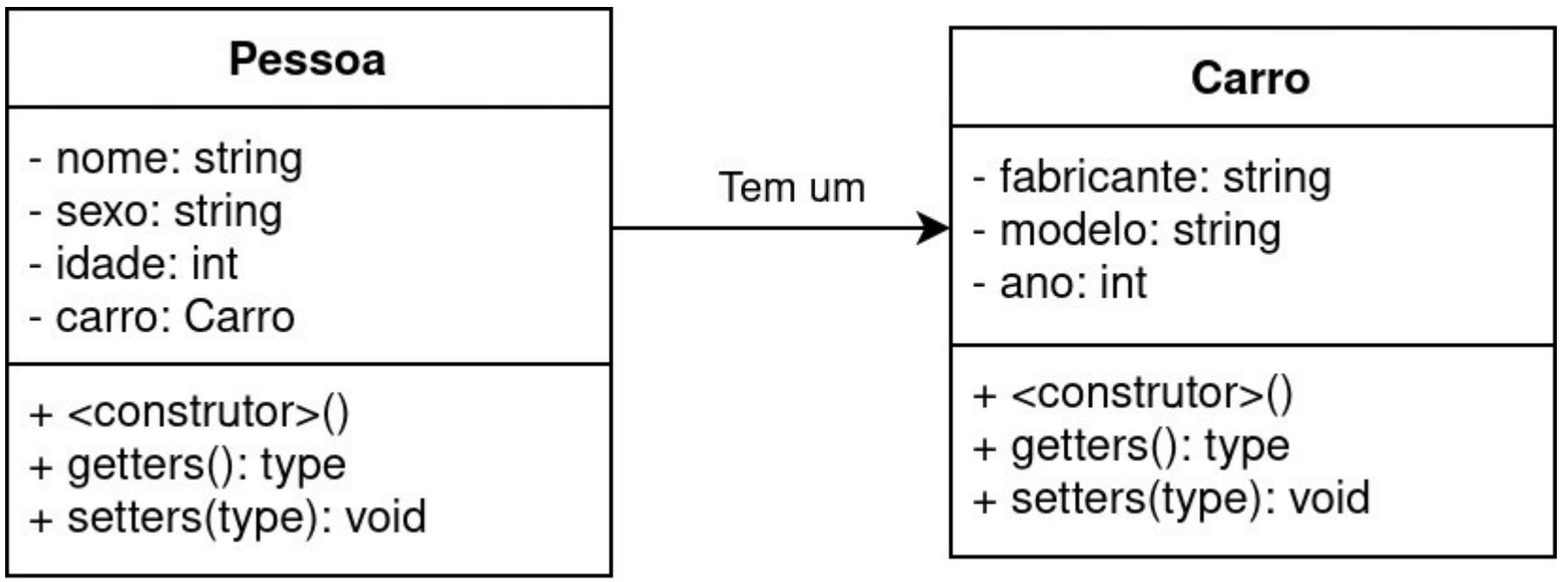
Pessoa

Tem um

Carro

Associação entre Classes

- Diagrama UML



Associação entre Classes

- Uma **pessoa** pode **não ter um carro**, da mesma forma que um **carro** pode **não pertencer** a **nenhuma pessoa**
- Mesmo assim, os **objetos** podem **existir de maneira independente** e funcionar perfeitamente
- Mas para que uma **pessoa** possa **ter** um **carro**, é necessário que **ambos objetos existam**, e que a **associação** entre eles seja **definida**
- Podemos **definir uma associação** de objetos através do **construtor** ou **método setter**

Exemplo

```python
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
```

```python
class Pessoa:

  def __init__(self, nome, sexo, idade, carro=None):
    self._nome = nome
    self._sexo = sexo
    self._idade = idade
    # Associação - Pessoa "tem um" Carro
    self._carro = carro

  def get_nome(self):
    return self._nome

  def get_sexo(self):
    return self._sexo

  def get_idade(self):
    return self._idade

  def get_carro(self):
    return self._carro
```

Exemplo

```
# Instancia um objeto "Carro"
carro = Carro('Ford', 'Bigode', 1938)

# Exibe os dados do carro
print('\nCarro:',
  carro.get_fabricante(),
  carro.get_modelo(),
  carro.get_ano())

# Instancia um objeto "Pessoa" sem "Carro"
pessoa = Pessoa('Dick Vigarista', 'Masculino', 42)

# Exibe os dados da pessoa
print('\nPessoa:',
  pessoa.get_nome(),
  pessoa.get_sexo(),
  pessoa.get_idade(),
  pessoa.get_carro())
```

Interface

- Pelos princípios do **encapsulamento** e **modularidade**, cada objeto é **autônomo** e responsável por **gerenciar seu próprios dados**
- Não é **prático**, e em alguns casos nem **seguro**, acessarmos **atributos** ou certos **métodos** de objetos associados
- Nesse caso, a **funcionalidade desejada** deve ser **implementada no objeto associado** e **invocada** através de um **método público**
- Chamamos de **interface** o conjunto de **métodos públicos** que os objetos usam para se **relacionarem**

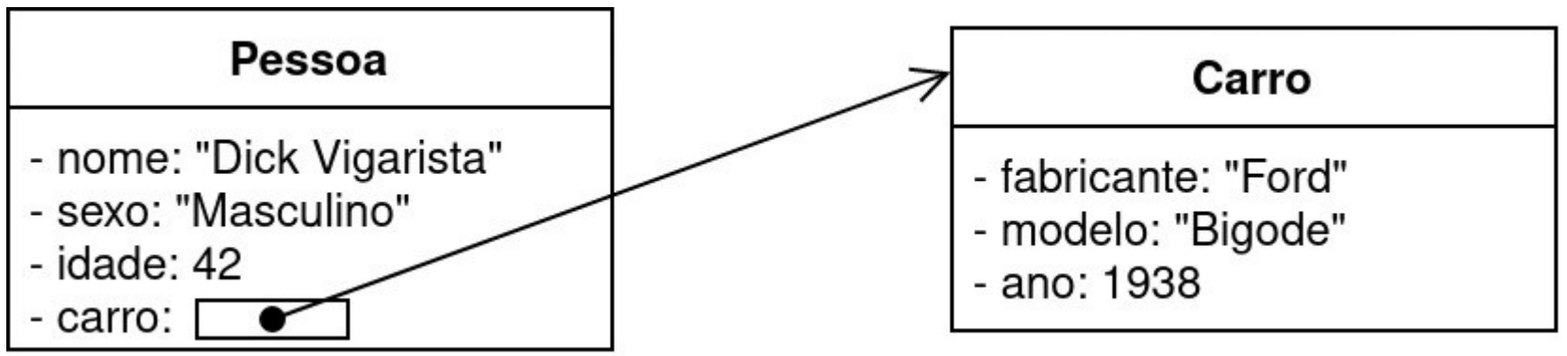
Exemplo

```
class Carro:

  def exibe_dados(self):
    # Exibe os dados do carro
    print('Carro:',
      self.get_fabricante(),
      self.get_modelo(),
      self.get_ano())


class Pessoa:

  def exibe_dados(self):
    # Exibe os dados da pessoa
    print('\nPessoa:',
      self.get_nome(),
      self.get_sexo(),
      self.get_idade())

    # Exibe os dados do carro da pessoa,
caso exista
    if self._carro:
      self._carro.exibe_dados()
    else:
      print('Carro: não definido')
```

```
# Instancia um objeto "Carro"
carro = Carro('Ford', 'Bigode', 1938)

# Exibe os dados do carro
carro.exibe_dados()

# Instancia um objeto "Pessoa" sem "Carro"
pessoa = Pessoa('Dick Vigarista', 'Masculino', 42)

# Exibe os dados da pessoa e do carro, caso exista
pessoa.exibe_dados()

# Associa o carro à pessoa
pessoa.set_carro(carro)

# Exibe os dados da pessoa e do carro
pessoa.exibe_dados()
```

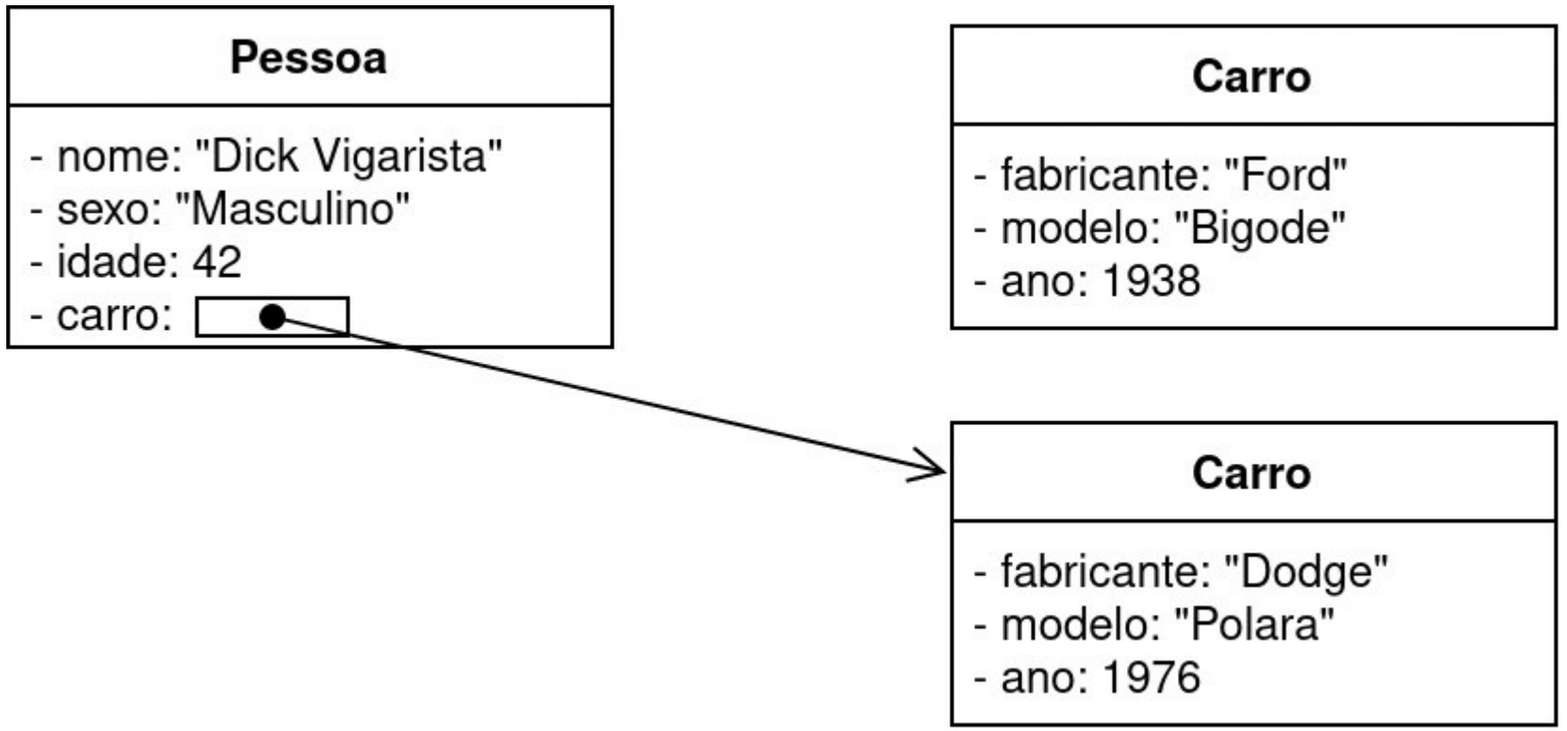
Objetos na Memória

- Visto que os objetos são **independentes** e criados **separadamente**, cada **instância** ocupa uma **região distinta na memória**
- Uma **associação** entre objetos armazena uma **referência** para o **endereço de memória** onde o **objeto está**, e não o **objeto propriamente dito** ou uma **cópia** dele



Objetos na Memória

- Para a pessoa “**trocar de carro**”, basta vincular **um outro objeto** previamente existente ao atributo carro
- O **carro anterior continua existindo**, porém, **não** está mais **vinculado** à pessoa



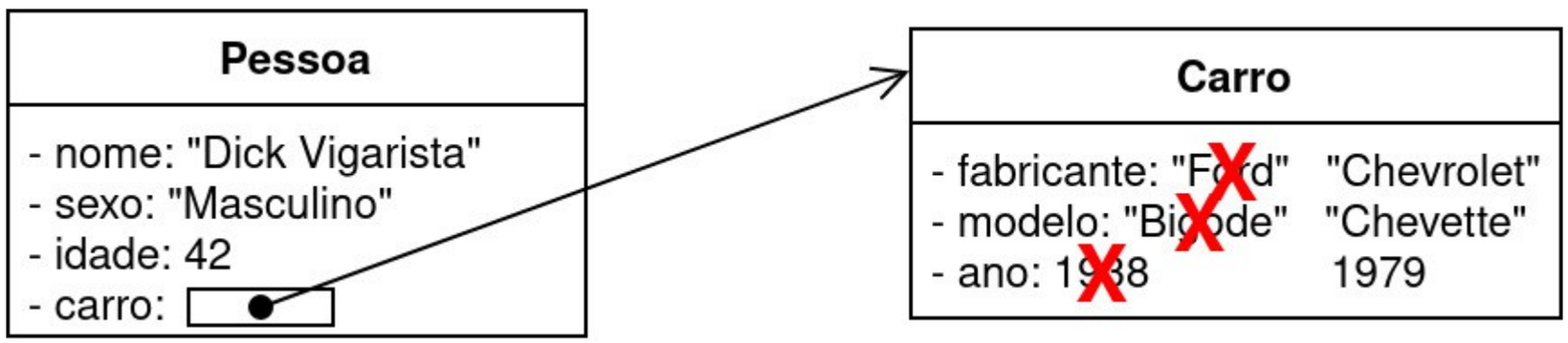
Exemplo

```
# Instancia um objeto "Carro"
carro_1 = Carro('Ford', 'Bigode', 1938)

# Instancia um objeto "Pessoa" com carro 1
pessoa = Pessoa('Dick Vigarista', 'Masculino', 42, carro_1)

# Exibe os dados da pessoa e do carro
pessoa.exibe_dados()

# Instancia outro objeto "Carro"
carro_2 = Carro('Dodge', 'Polara', 1976)

# Associa o carro à pessoa
pessoa.set_carro(carro_2)

# Exibe os dados da pessoa e do carro
pessoa.exibe_dados()

# O carro_1 continua existindo
carro_1.exibe_dados()
```

Objetos na Memória

- Se **modificarmos** qualquer atributo de um objeto, as **alterações refletem instantaneamente** para todas as **associações** feitas **àquele objeto**

- Isso comprova que os objetos são **independentes** e as **associações** armazenam **apontamentos** entre eles, e **não** guardam **cópias** dos **dados**



Exemplo

```
# Instancia um objeto "Carro"
carro = Carro('Ford', 'Bigode', 1938)

# Instancia um objeto "Pessoa" com "Carro"
pessoa = Pessoa('Dick Vigarista', 'Masculino', 42, carro)

# Exibe os dados da pessoa e do carro
pessoa.exibe_dados()

# Altera dados do carro diretamente no objeto
carro.set_fabricante('Chevrolet')
carro.set_modelo('Chevette')

# Altera dados do carro através da associação
pessoa.get_carro().set_ano(1979)

# Exibe os dados da pessoa e do carro
pessoa.exibe_dados()
```

Atividade

- No Canvas