Programação Orientada a Objetos

Prof. Márcio Miguel Gomes

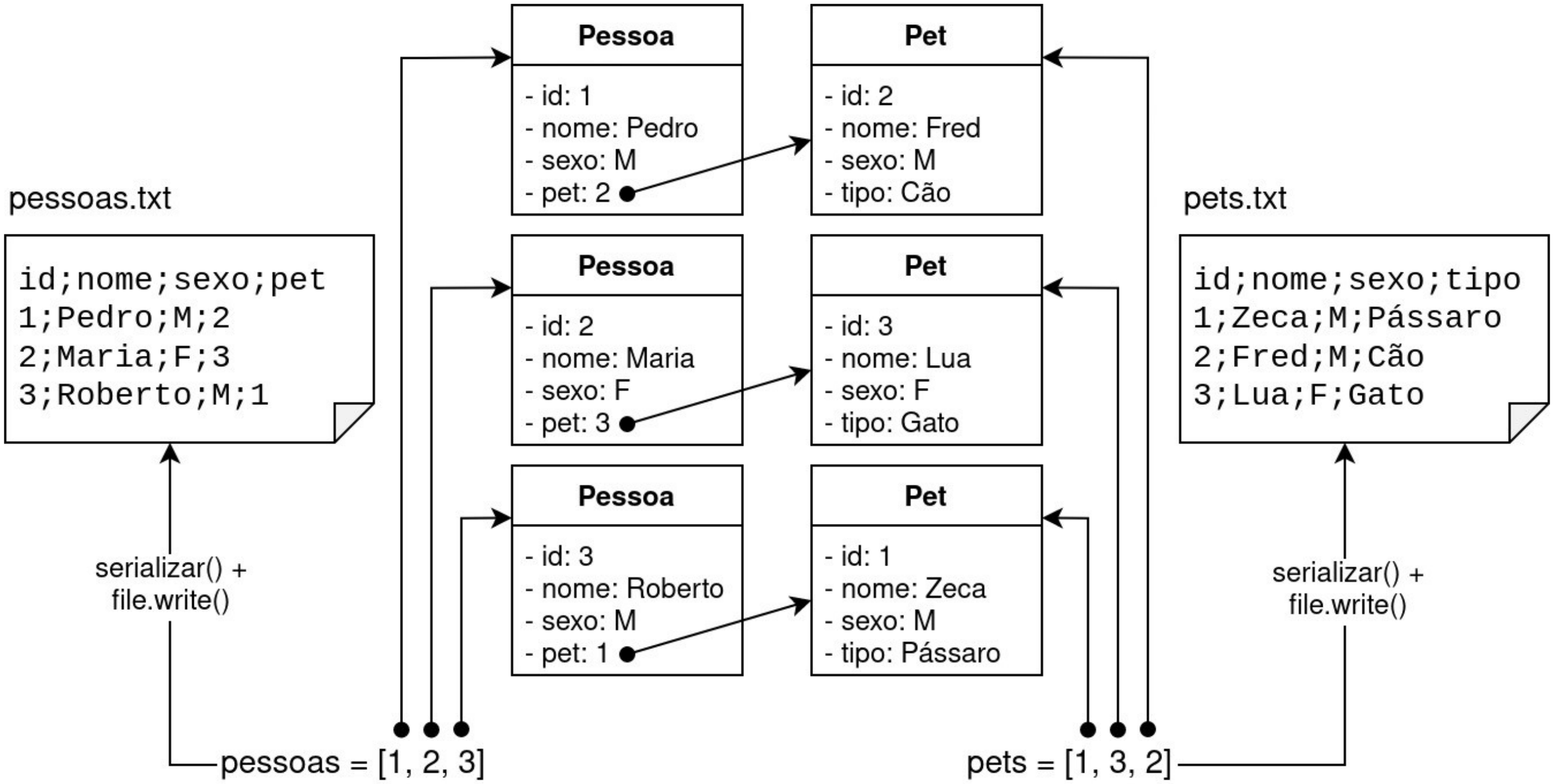
Persistência de Dados

- Na aula anterior, vimos como trabalhar com persistência de objetos contendo atributos simples, com tipos de dados primitivos
- Mas como fazer quando os objetos possuem atributos que são outros objetos?
- Como deve ser o processo de gravação?
- Como deve ser a estrutura do arquivo de dados?
- Como deve ser o processo de leitura?
- Como manter a associação entre os objetos?

Gravação

- Cada objeto deve ter um identificador único por classe, que normalmente é um numero inteiro sequencial
- Para cada classe, deve haver uma variável “container” para armazenar uma coleção dos objetos dessa classe
- O método “serializar” continua retornando os valores dos atributos primitivos utilizando um separador padrão
- Porém, para os atributos que são objetos, o método deve retornar o identificador único do objeto
- Então, para cada objeto que se quer salvar, deve-se chamar o método de “serialização” e gravar seus dados no arquivo

Gravação



Gravação

- Exemplo de algoritmo para salvar objetos:

1. Para cada “container” de objetos:
2. Abrir o arquivo para armazenar os objetos
3. Adicionar um cabeçalho, caso necessário
4. Para cada objeto do “container”:
5. Chamar a função de “serialização”
6. Adicionar o objeto serializado no arquivo
7. Fechar o arquivo

Gravação

```
class Pessoa:
  def __init__(self, id, nome, sexo, pet):
    self._id = id
    self._nome = nome
    self._sexo = sexo
    self._pet = pet

  def serializar(self):
    return f'\n{self._id};{self._nome};{self._sexo};{self._pet.get_id()}'

class Pet:
  def __init__(self, id, nome, sexo, tipo):
    self._id = id
    self._nome = nome
    self._sexo = sexo
    self._tipo = tipo

  def get_id(self):
    return self._id

  def serializar(self):
    return f'\n{self._id};{self._nome};{self._sexo};{self._tipo}'
```

Gravação

```
if __name__ == '__main__':
  pets = []
  pessoas = []

  pets.append(Pet(1, 'Zeca', 'M', 'Pássaro'))
  pets.append(Pet(2, 'Fred', 'M', 'Cão'))
  pets.append(Pet(3, 'Lua', 'F', 'Gato'))

  pessoas.append(Pessoa(1, 'Pedro', 'M', pets[1]))
  pessoas.append(Pessoa(2, 'Maria', 'F', pets[2]))
  pessoas.append(Pessoa(3, 'Roberto', 'M', pets[0]))

  arq = open('pessoas.txt', 'w')
  arq.write('id;nome;sexo;pet')     # Escreve o cabeçalho do arquivo

  for pessoa in pessoas:
    arq.write(pessoa.serializar())  # Escreve os dados das pessoas
  arq.close()

  arq = open('pets.txt', 'w')
  arq.write('id;nome;sexo;tipo')    # Escreve o cabeçalho do arquivo
  for pet in pets:
    arq.write(pet.serializar())     # Escreve os dados dos pets
  arq.close()
```

Leitura

- O método “deserializar” continua recebendo uma *string* com os dados serializados, analisa, identifica e separa cada dado, e depois atualiza os atributos primitivos do objeto
- Porém, para os atributos que são objetos, o método deve procurar o objeto no respectivo “container” através de seu identificador único, e retornar uma referência para o objeto localizado
- Então, para cada objeto que se quer recuperar, deve-se ler uma linha do arquivo, instanciar dinamicamente o objeto e chamar o método de “deserialização”
- Dica: O próprio método construtor pode disparar a deserialização

Leitura

- Exemplo de algoritmo para recuperar objetos:

1. Para cada arquivo de objetos:
2. Abrir o arquivo contendo os objetos
3. Descartar o cabeçalho, caso exista
4. Para cada linha do arquivo:
5. Instanciar dinamicamente um objeto e armazená-lo no “container”
6. Chamar a função “deserializar” passando a linha como parâmetro
7. Para atributos complexos, localizar pelo “id” e retornar uma referência
8. Fechar o arquivo

Leitura

```
class Pessoa:
  def __init__(self, linha):
    self.deserializar(linha)

  def deserializar(self, linha):
    dados = linha.split(';')
    self._id = int(dados[0])
    self._nome = dados[1]
    self._sexo = dados[2]
    self._pet = localiza_pet(int(dados[3]))

  def exibe_dados(self):
    print('* Pessoa *')
    print('Id:', self._id)
    print('Nome:', self._nome)
    print('Sexo:', self._sexo)
    print('* Pet *')
    if self._pet:
      self._pet.exibe_dados()
    else:
      print('Desconhecido')
    print()
```

Leitura

```
class Pet:
  def __init__(self, linha):
    self.deserializar(linha)

  def deserializar(self, linha):
    dados = linha.split(';')
    self._id = int(dados[0])
    self._nome = dados[1]
    self._sexo = dados[2]
    self._tipo = dados[3]

  def get_id(self):
    return self._id

  def exibe_dados(self):
    print('Id:', self._id)
    print('Nome:', self._nome)
    print('Sexo:', self._sexo)
    print('Tipo:', self._tipo)
```

Leitura

```
def localiza_pet(id):
  for pet in pets:
    if pet.get_id() == id:
      return pet
  return None

if __name__ == '__main__':
  pets = []
  pessoas = []

  arq = open('pets.txt')
  arq.readline()              # Descarta o cabeçalho do arquivo
  for linha in arq:           # Instancia os objetos
    pets.append(Pet(linha.strip()))
  arq.close()

  arq = open('pessoas.txt')
  arq.readline()              # Descarta o cabeçalho do arquivo
  for linha in arq:           # Instancia os objetos
    pessoas.append(Pessoa(linha.strip()))
  arq.close()

  for pessoa in pessoas:
    pessoa.exibe_dados()    # Exibe os dados dos objetos
```

Atividade

- Modifique o código estudado em aula para que uma *pessoa* possa ter diversos *pets*. Para testar a modificação, faça um programa com o seguinte menu:
  1. **Cadastrar pet**: pede todos os dados de um *pet*, exceto o “id”. Cria o objeto e o armazena na coleção “pets”. O “id” deve ser gerado automaticamente
  2. **Cadastrar pessoa**: pede todos os dados de uma *pessoa*, exceto o “id”. Cria o objeto e o armazena na coleção “pessoas”. O “id” deve ser gerado automaticamente
  3. **Adotar**: solicita o nome da *pessoa* e do *pet*, e os localiza nas respectivas coleções. Caso ambos existam, realiza a associação entre eles
  4. **Listar tudo**: mostra todos os dados das *pessoas* associadas a seus respectivos *pets*
  5. **Listar por tipo de pet**: pede para o usuário informar um *tipo* de *pet* e exibe apenas o *nome* das *pessoas* e de seus respectivos *pets* do *tipo* informado
  6. **Salvar**: salva nos arquivos todos os objetos que estão em memória
  7. **Carregar**: carrega para a memória os objetos que estão nos arquivos
  8. **Sair**: encerra o programa

Observação: Os arquivos devem ter um cabeçalho com o nome de cada atributo