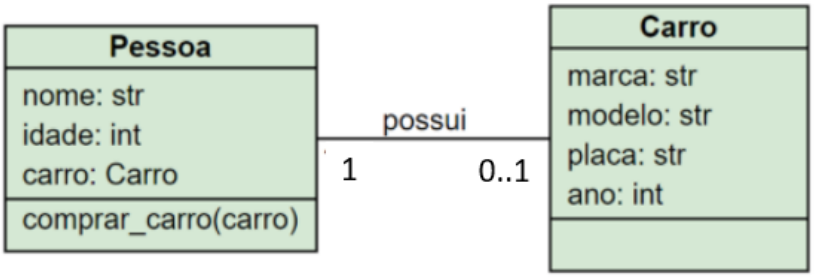
Faculdade IMPACTA	**Programação Orientada a Objetos** **Relacionamento entre Classes** **Exercícios**

Exercício 01

A associação entre classes ocorre quando uma classe possui atributos que são objetos de outra classe. De acordo com esse conceito, observe o diagrama de classes abaixo, que representa uma associação onde uma Pessoa possui um Carro.



Implemente as classes Pessoa e Carro.

Classe Pessoa:

Atributos:

- **nome**: nome da pessoa
- **idade**: idade da pessoa
- **carro**: objeto da classe Carro (valor inicial definido no construtor como *None*)

Métodos:

- **comprar_carro**: recebe um objeto Carro e atribui esse objeto ao atributo carro.

Classe Carro:

Atributos:

- **marca**: marca do carro
- **modelo**: modelo do carro
- **placa**: placa do carro
- **ano**: ano de fabricação do carro

Métodos:

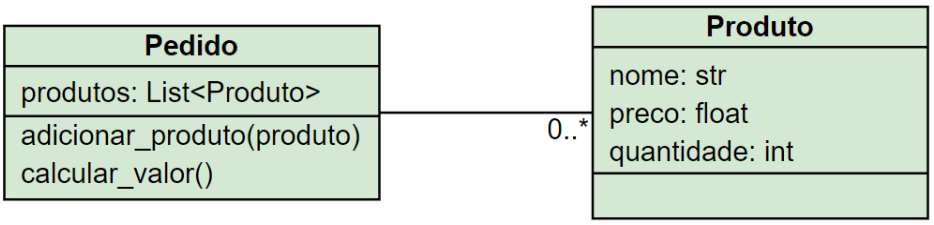
- Não possui

Você pode utilizar o trecho de programa abaixo para testar as suas classes:

```
meucarro = Carro('Volkswagen', 'Gol', 'AAA-1111', 2015)
eu = Pessoa('João', 25)
eu.comprar_carro(meucarro)
print('Meu nome: ', eu.nome)                          # imprime: João
print('Modelo do meu carro: ', eu.carro.modelo)       # imprime :Gol
print('Placa do meu carro: ', eu.carro.placa)         # imprime: AAA-1111
```

Exercício 02

Observe o diagrama de classes abaixo, que representa uma associação onde um Pedido possui uma lista de Produtos.



Implemente as classes Produto e Pedido.

Classe Pedido:

Atributos:

- **produtos**: Lista de objetos do classe Produto (inicializar no construtor como uma lista vazia)

Métodos:

- **adicionar_produto**: recebe um objeto Produto e o adiciona na lista de produtos.
- **calcular_valor**: deve retornar o valor total do pedido (soma dos preços de todos os produtos do pedido)

Classe Produto:

Atributos:

- **nome**: nome do produto
- **preco**: preço do produto
- **quantidade**: quantidade de itens do produto

Métodos:

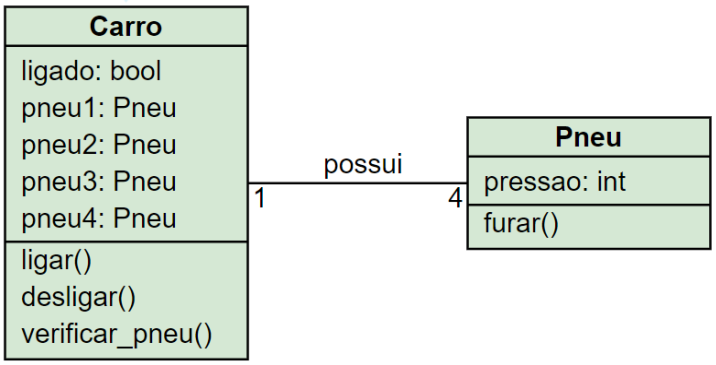
- Não possui

Você pode utilizar o programa abaixo para testar as suas classes:

```
cafe = Produto('Café Solúvel', 5.50, 1)
arroz = Produto('Arroz Integral', 4.90, 2)
feijao = Produto('Feijão Preto', 2.80, 2)
meu_pedido = Pedido()
meu_pedido.adicionar_produto(cafe)
meu_pedido.adicionar_produto(arroz)
meu_pedido.adicionar_produto(feijao)
print('O valor total é: ', meu_pedido.calcular_valor())        # imprime 20.90
```

Exercício 03

Observe o diagrama de classes abaixo e implemente as classes Carro e Pneu. O diagrama representa uma associação onde um Carro possui quatro Pneus.



Classe Carro:

Atributos:

- **ligado**: valor booleano que indica se o carro está ligado ou desligado (definido no construtor com o valor False)
- **pneu1**, **pneu2**, **pneu3**, **pneu4**: objetos do tipo Pneu que representam cada um dos pneus do carro.

Métodos:

- **ligar**: altera o valor do atributo ligado para True
- **desligar**: altera o valor do atributo ligado para False
- **verificar_pneu**: se o carro estiver ligado, esse método deve exibir a pressão de cada um dos pneus. Se o carro estiver desligado, o método deve exibir a mensagem 'Carro Desligado'

Classe Pneu:

Atributos:

- **pressao**: valor inteiro que indica a pressão de ar do pneu

Métodos:

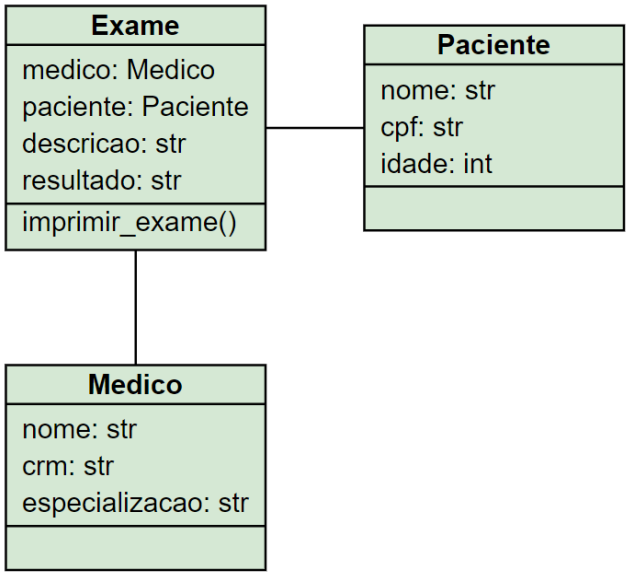
- **furar()**: simula um pneu furado, alterando o valor do atributo pressao para zero.

Você pode utilizar o trecho de programa abaixo para testar as suas classes.

```
p1 = Pneu(32)
p2 = Pneu(32)
p3 = Pneu(36)
p4 = Pneu(36)
meucarro = Carro(p1, p2, p3, p4)
meucarro.ligar()
meucarro.pneu3.furar()
meucarro.verificar_pneu()        # Deve exibir a pressão de cada pneu.
meucarro.desligar()
meucarro.verificar_pneu()        # Deve exibir mensagem que o carro está desligado
```

Exercício 04

Implemente o diagrama de classes abaixo.



Classe Paciente

Atributos:

- **nome**
- **cpf**
- **idade**

Métodos:

- não possui

Classe Medico

Atributos:

- **nome**
- **crm**
- **especializacao**

Métodos:

- não possui

Classe Exame

Atributos:

- **medico**: objeto da classe Medico
- **paciente**: objeto da classe Paciente
- **descricao**
- **resultado**

Métodos:

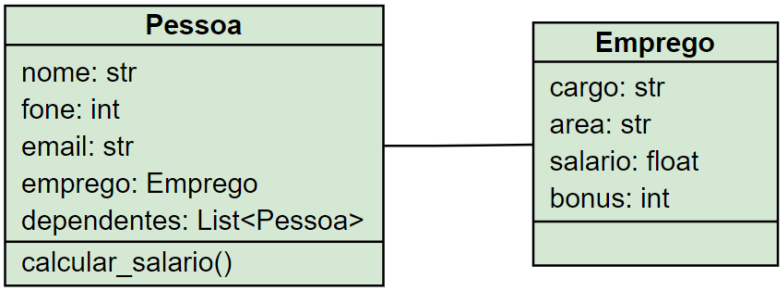
- **imprimir_exame()**: exibe um relatório com os dados do exame (inclusive os dados do médico e do paciente)

Você pode utilizar o programa a seguir para testar as suas classes:

```
paciente = Paciente('Marcelo Silva', '033444555-22', 26)
medico = Medico('Ana Beatriz', 333431, 'Clínico Geral')
exame01 = Exame(medico, paciente, 'COVID-19', 'Negativo')
exame01.imprimir_exame()
# Deve exibir relatório com os dados do exame (inclusive os do médico e do paciente)
```

Exercício 05

Implemente o diagrama de classes abaixo.



Classe Emprego

Atributos:

- **cargo**
- **area**
- **salario**
- **bonus**: percentual de bonificação a ser acrescentado ao salário do funcionário, de acordo com a quantidade de dependentes. Por exemplo, se o bônus for de 2%, e o funcionário tiver 3 dependentes, ele receberá 6% de acréscimo sobre o salário.

Métodos:

- não possui

Classe Pessoa

Atributos:

- **nome**
- **fone**
- **email**
- **emprego**: objeto do tipo Emprego
- **dependentes**: lista de objetos do tipo Pessoa

Métodos:

- **calcular_salario**: retorna o valor do salário do funcionário, de acordo com o percentual de bonificação e quantidade de dependentes.

Você pode utilizar o programa a seguir para testar as suas classes:

```
emprego = Emprego("Programador", "TI", 1000, 5)
pessoa1 = Pessoa("Paulo", "11-99999999", "paulo@email.com", emprego)

# dois dependentes (o dependente também é um objeto Pessoa)
dep1 = Pessoa("Maria", "", "", None)
dep2 = Pessoa("Joao", "", "", None)

# adiciona dependentes na lista de dependentes da pessoa1
pessoa1.dependentes.append(dep1)
pessoa1.dependentes.append(dep2)

print("Salario: ", pessoa1.calcular_salario())               # imprime 1100.0
```