

# COMPOSIÇÃO E ASSOCIAÇÃO ENTRE OBJETOS

IFBA – Campus Ilhéus  
Curso: Back End com Python

# Objetivo da Aula

- Apresentar como criar relações entre objetos para modelar sistemas complexos.

# Por que precisamos relacionar objetos?

- Em sistemas reais, objetos cooperam entre si, por exemplo:
  - Um Aluno pertence a um Curso
  - Um Carro possui um Motor
- Em Programação Orientada a Objetos, **nenhum objeto trabalha sozinho.**

# Associação

- Associação é quando um objeto depende do outro, mas ambos podem existir separadamente.
- Exemplos da vida real:
  - Professor ↔ Escola
  - Cliente ↔ Pedido
  - Pessoa ↔ Conta bancária

# Associação

- Um objeto usa outro objeto, mas cada um pode existir separadamente.
  - Relação fraca
  - Objetos têm vida independente
- Exemplo
  - **Turma** → usa **Professor**  
Mas ambos podem existir sem o outro.

# Associação - Exemplo

```
class Professor:
```

```
    def __init__(self, nome):  
        self.nome = nome
```

```
class Turma:
```

```
    def __init__(self, codigo, professor):  
        self.codigo = codigo  
        self.professor = professor # associação (apenas  
referência)
```

# Associação - Exemplo - Continuação

```
# Programa principal  
prof = Professor("Carlos")  
turma = Turma("T01", prof)  
  
print(turma.professor.nome)
```

# Tipos de associação

Tipo	Exemplo	Quem conhece quem?
Unidirecional	Turma → Professor	Só Turma conhece Professor
Bidirecional	Aluno ↔ Professor	Ambos sabem um do outro
1→1	Pessoa → CNH	Um para um
1→*	Professor → Alunos	Um para muitos
↔	Alunos ↔ Disciplinas	Muitos para muitos

# Composição

- Um objeto é parte de outro e não pode existir sem ele.
  - Relação forte
  - O todo gerencia o ciclo de vida da parte
- Exemplo real:
  - Carro → Motor  
Se o carro deixa de existir, o motor também

# Composição - Exemplo

```
class Motor:  
    def __init__(self, potencia):  
        self.potencia = potencia  
  
class Carro:  
    def __init__(self, modelo, potencia_motor):  
        self.modelo = modelo  
        self.motor = Motor(potencia_motor) #  
composição (parte-todo)
```

# Composição - Exemplo

```
# Programa principal  
carro = Carro("Corolla", 2.0)  
print(carro.motor.potencia)
```

- motor é criado dentro do Carro
- O motor não existe fora dele

# Comparação Geral

Conceito	Associação	Composição
Vida independente?	Sim	Não
Grau de dependência	Baixo	Alto
Exemplo	Turma → Professor	Carro → Motor
Ideal pra...	Relações flexíveis	Relações parte–todo

# Resumo

- Se o objeto principal deixar de existir e o outro não faz sentido sozinho → composição
- Se cada um pode viver por conta própria → associação

# Exercício 1

- Marque **A** para Associação e **C** para Composição:

Relação	A / C
Pessoa → Endereço	<u>A</u>
Casa → Cômodos	<u>C</u>
Escola → Alunos	<u>A</u>
Livro → Páginas	<u>C</u>
Computador → Placa-mãe	<u>C</u>

# Exercício 2

## ◎ Faça um trecho de código

a) Modele a relação **Livro (titulo, autor)** e **Autor**

- Qual tipo de relação?
- Mostre apenas o atributo que representa essa relação

# Exercício 2 - Solução

- Associação: Autor existe mesmo sem o livro.

```
class Livro:
```

```
    def __init__(self, titulo, autor):  
        self.titulo = titulo  
        self.autor = autor # associação
```

# Exercício 3

## ◎ Faça um trecho de código

a) Modele a relação **Casa (porta)** e **Porta (material, cor)**

- Qual tipo?
- Mostre apenas o construtor da Casa

# Exercício 3 - Solução

- Composição: Porta criada dentro da Casa  
→ depende dela

class Porta:

```
def __init__(self, material, cor):  
    self.material = material  
    self.cor = cor  
    self.aberta = False # começa fechada
```

def abrir(self):

```
    self.aberta = True  
    print("A porta foi aberta.")
```

# Exercício 3 – Solução - Continuação

- Composição: Porta criada dentro da Casa → depende dela

```
def fechar(self):  
    self.aberta = False  
    print("A porta foi fechada.")
```

```
class Casa:  
    def __init__(self):  
        self.porta = Porta() # composição
```

# Exercício 4

- Implemente um sistema básico para um consultório médico com as seguintes classes:
  - Classe Paciente com os atributos nome e idade e o método mostrar\_informacoes(), que exibe os dados do paciente.
  - Classe Medico com os atributos nome e especialidade e o método mostrar\_informacoes(), que exibe os dados do médico.

# Exercício 4 - Continuação

- Classe Consulta, com os atributos:
  - paciente (objeto Paciente)
  - medico (objeto Medico)
  - data (str)
  - motivo (str)
- Método:
  - detalhes(): exibe todas as informações da consulta
- Aqui temos **associação**, pois a consulta *usa* objetos de médico e paciente:

# Exercício 4 - Continuação

- Programa principal:

- Crie 2 pacientes
- Crie 1 médico
- Crie 2 consultas com esses objetos
- Exiba os detalhes das consultas usando detalhes()

# Exercício 4 – Solução

```
class Paciente:  
    def __init__(self, nome, idade):  
        self.nome = nome  
        self.idade = idade  
  
    def mostrar_informacoes(self):  
        print(f"paciente: {self.nome}, Idade:  
{self.idade} anos")
```

# Exercício 4 - Solução - Continuação

```
class Medico:
```

```
    def __init__(self, nome, especialidade):
```

```
        self.nome = nome
```

```
        self.especialidade = especialidade
```

```
    def mostrar_informacoes(self):
```

```
        print(f"Médico: Dr(a). {self.nome},")
```

```
        Especialidade: {self.especialidade}")
```

# Exercício 4 - Solução - Continuação

class Consulta:

```
def __init__(self, paciente, medico, data,  
motivo):  
    self.paciente = paciente # associação  
    self.medico = medico      # associação  
    self.data = data  
    self.motivo = motivo
```

# Exercício 4 – Solução - Continuação

```
def detalhes(self):  
    print("\n--- Detalhes da Consulta ---")  
    print(f"Data: {self.data}")  
    print(f"Motivo: {self.motivo}")  
    print(f"paciente: {self.paciente.nome}")  
    print(f"Médico: Dr(a). {self.medico.nome}  
({self.medico.especialidade})")  
    print("-----")
```

# Exercício 4 - Solução - Continuação

```
# Programa Principal
```

```
p1 = Paciente("Ana", 30)
```

```
p2 = Paciente("Carlos", 45)
```

```
m1 = Medico("Mariana", "Cardiologia")
```

```
c1 = Consulta(p1, m1, "12/10/2025", "Check-up")
```

```
c2 = Consulta(p2, m1, "15/10/2025", "Dor no peito")
```

```
c1.detalhes()
```

```
c2.detalhes()
```

# Exercício 5

- Um computador possui um Processador. Se o computador deixar de existir, o processador também deixa, ou seja, composição.
- Classe Processador, com os atributos:
  - marca, velocidade
- Classe Computador, com os atributos:
  - modelo, marca\_proc, velocidade\_proc
  - Método exibir\_dados, que exibe os três dados de computador.
  - No PP crie um computador(“Dell Inspiron”, “Intel Core i5”, 3.2) e exiba os dados com o método implementado.

# Exercício 5 - solução

```
class Processador:  
    def __init__(self, marca, velocidade):  
        self.marca = marca  
        self.velocidade = velocidade  
  
class Computador:  
    def __init__(self, modelo, marca_proc,  
velocidade_proc):  
        self.modelo = modelo  
        self.processador =  
Processador(marca_proc, velocidade_proc)
```

# Exercício 5 – solução - continuação

```
def exibir_dados(self):
    print(f"Modelo: {self.modelo}")
    print(f"Processador:
{self.processador.marca} com velocidade
{self.processador.velocidade}")

comp = Computador("Dell Inspiron", "Intel
Core i5", 3.2)
comp.exibir_dados()
```