Módulo 8: Backend com C# e Python

Dia 04: Orientação a Objetos

João Quentino

29 de dezembro de 2025

Roteiro do Dia 04

Objetivo da Aula

Ao final desta aula, você será capaz de:

- Criar classes com atributos e métodos em Python e C#.
- Usar encapsulamento e *properties* em C#.
- Aplicar herança e polimorfismo em exemplos simples.
- Entender, em alto nível, o conceito de Injeção de Dependência (DI).
- Modelar classes de um pequeno e-commerce.

Por que Orientação a Objetos?

- Organizar o código como se fosse o **mundo real**: produtos, clientes, pedidos.
- Facilitar a manutenção e a evolução do sistema.
- Reaproveitar código com herança e composição.
- Preparar a base para arquiteturas maiores (APIs, serviços, etc.).

Classe simples em Python

Classe Produto

```
class Produto:
    def __init__(self, nome, preco):
        self.nome = nome
        self.preco = preco

    def exibir(self):
        print(f"{self.nome} - R$ {self.preco}")


produto = Produto("Teclado", 150.0)
produto.exibir()
```

Classe simples em C#

Classe Produto

```
public class Produto
{
    public string Nome { get; set; } = string.Empty;
    public decimal Preco { get; set; }

    public void Exibir()
    {
        Console.WriteLine($"{Nome} - R$ {Preco}");
    }
}
```

Encapsulamento em C#

Controlando acesso

```
public class Produto
{
    public string Nome { get; private set; }
    public decimal Preco { get; private set; }

    public Produto(string nome, decimal preco)
    {
        Nome = nome;
        Preco = preco;
    }

    public void AumentarPreco(decimal percentual)
    {
        Preco = Preco * (1 + percentual);
    }
}
```

Herança em Python

Produto físico vs digital

```
class Produto:
    def __init__(self, nome, preco):
        self.nome = nome
        self.preco = preco

class ProdutoDigital(Produto):
    def __init__(self, nome, preco, tamanho_mb):
        super().__init__(nome, preco)
        self.tamanho_mb = tamanho_mb
```

Herança e polimorfismo em C#

Classe base e derivada

```
public class Produto
{
    public string Nome { get; set; } = string.Empty;
    public decimal Preco { get; set; }
    public virtual decimal CalcularTotal()
    {
        return Preco;
    }
}

public class ProdutoComDesconto : Produto
{
    public decimal Desconto { get; set; }
    public override decimal CalcularTotal()
    {
        return Preco - Desconto;
    }
}
```

O que é Injeção de Dependência?

- Uma forma de **entregar** para uma classe o que ela precisa (dependências).
- Em vez da classe criar tudo sozinha, alguém de fora fornece as instâncias.
- Facilita testes, troca de implementações e organização.

Exemplo de ideia:

- `ServicoDePedido` recebe uma `interface IRepositorioPedido`.
- Em produção, você usa um repositório real com banco.
- Em teste, um repositório falso em memória.

Desafio Guiado: Modelar Classes

Contexto

Vamos modelar as classes principais de um pequeno e-commerce.

Entidades sugeridas:

- `Produto`
- `Cliente`
- `Pedido`
- `ItemPedido`

Python (Visão Geral do Modelo)

Estrutura simplificada

```
class Cliente:
    def __init__(self, nome, email):
        self.nome = nome
        self.email = email

class Pedido:
    def __init__(self, cliente):
        self.cliente = cliente
        self.itens = []
```

C# (Visão Geral do Modelo)

Estrutura simplificada

```
public class Cliente
{
    public string Nome { get; set; } = string.Empty;
    public string Email { get; set; } = string.Empty;
}

public class Pedido
{
    public Cliente Cliente { get; set; } = new Cliente
        ();
    public List<ItemPedido> Itens { get; set; } = new
        List<ItemPedido>();
}
```

Resumo do Dia 04

- Classes, atributos e métodos em Python e C#.
- Encapsulamento e uso de *properties* em C#.
- Herança e polimorfismo com exemplos simples.
- Noção de Injeção de Dependência.
- Modelagem inicial de um e-commerce.

Próximos Passos

Vamos modelar mais!

- Adicionar regras de negócio (ex: não permitir quantidade negativa).
- Calcular o total do pedido com base nos itens.
- Pensar como essas classes virariam endpoints de uma API.