



INTRODUÇÃO À PROGRAMAÇÃO ORIENTADA A OBJETOS COM C#

Conceitos de Encapsulamento, Herança
e Polimorfismo

Apresentado por Gustavo Campos



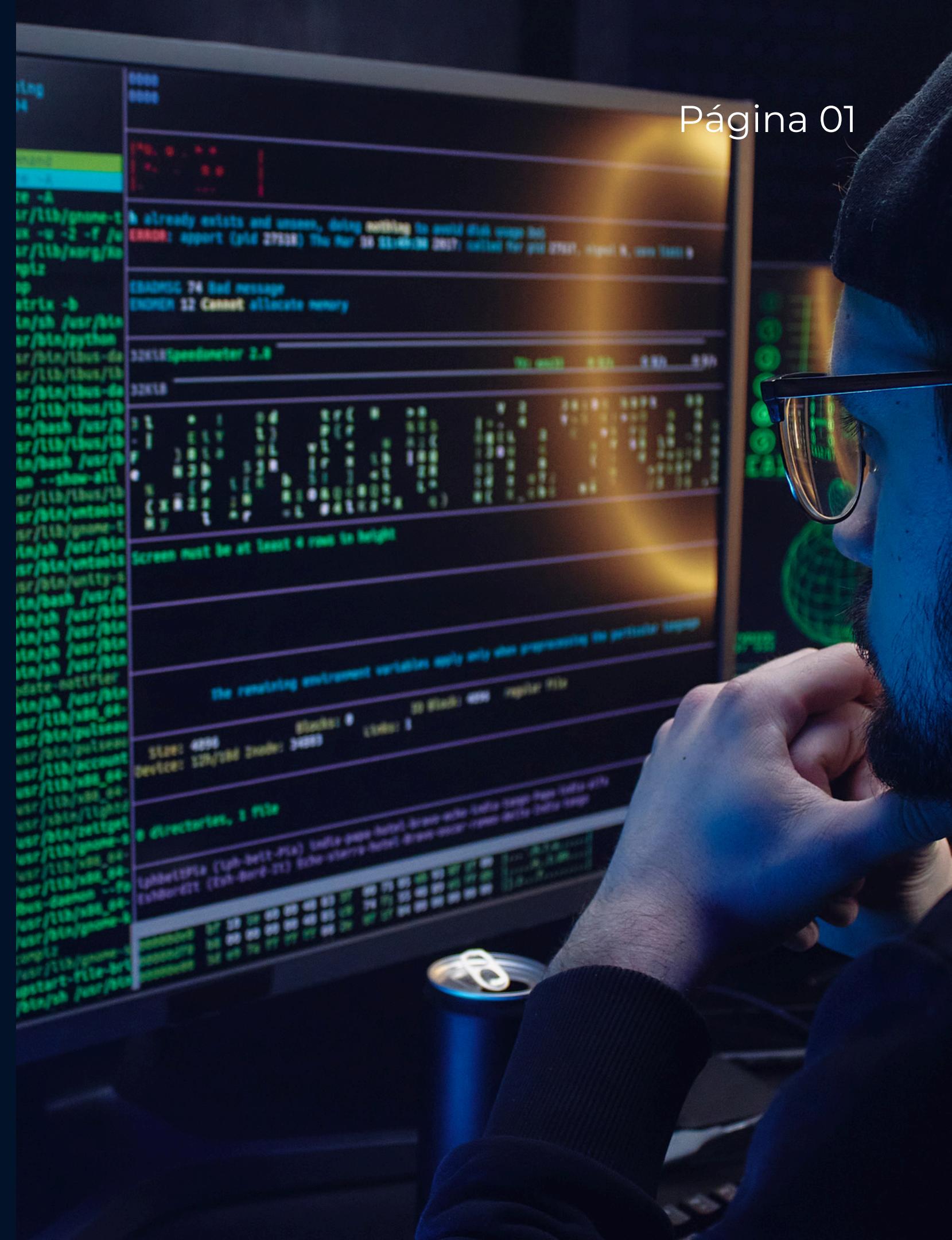


O QUE É PROGRAMAÇÃO ORIENTADA A OBJETOS (POO)?

A Programação Orientada a Objetos (POO) é um paradigma de programação que organiza o código em "objetos". Esses objetos são entidades que combinam dados (propriedades) e comportamentos (métodos) em uma única estrutura.

O objetivo da POO é facilitar a criação de programas complexos ao modelar o mundo real por meio de objetos. Ela permite que os desenvolvedores reutilizem código, criem sistemas modulares e mantenham o controle sobre os dados.

Explicando com um exemplo: um carro em um jogo pode ser um objeto com características como "cor" e "modelo", e comportamentos como "acelerar" e "frear".





ELEMENTOS DA POO

Página 02

- *Classes: São como plantas ou moldes que definem o que um objeto pode fazer e quais informações ele pode conter. Uma classe serve como um "modelo" para a criação de objetos.*
- *Objetos: São instâncias de classes. Um objeto é criado a partir de uma classe, ou seja, é um exemplo concreto de uma classe em uso.*
- *Atributos (Propriedades): São as características de um objeto. No exemplo do carro, atributos podem ser "cor", "marca" ou "ano".*
- *Métodos: São as ações ou comportamentos que um objeto pode realizar.*

Exemplo:

```
class Carro {  
    public string cor; // Aqui é atributo  
    public void Acelerar() { // Aqui é método  
        // Aqui é onde vai o código do método  
    } }
```



CLASSE EM C#

Uma classe é um bloco de construção fundamental em C# para a POO. Ela serve como um plano ou esqueleto para a criação de objetos. Dentro de uma classe, você define as propriedades (atributos) e as funções (métodos) que seus objetos terão.

Exemplo:

```
class Pessoa {  
    public string nome;  
    public int idade;  
  
    public void Falar() {  
        Console.WriteLine("Olá, eu sou " + nome);  
    }  
}
```

Aqui, Pessoa é uma classe que tem dois atributos (nome e idade) e um método (Falar()), que exibe uma mensagem no console.



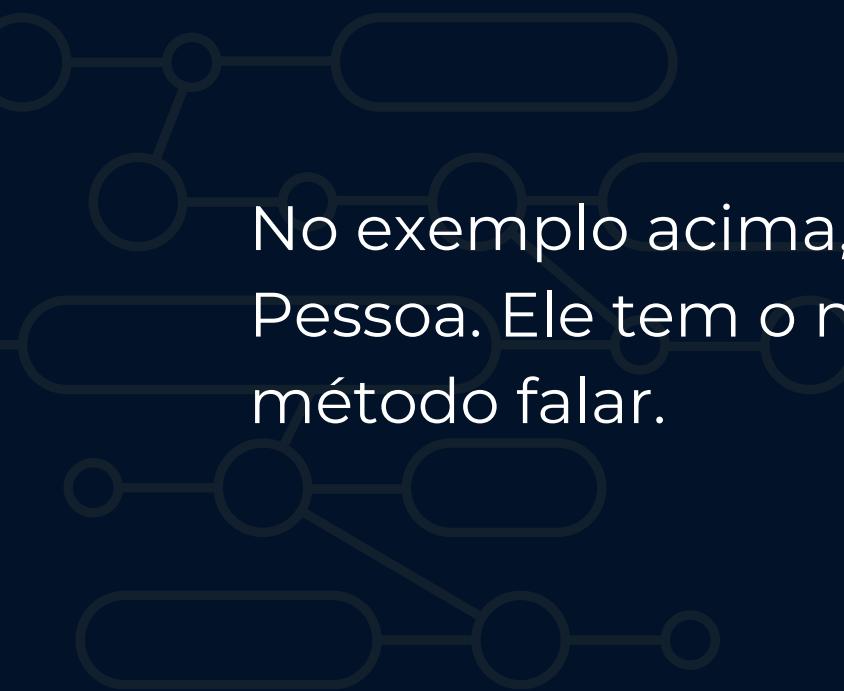


OBJETO

Página 04

Quando você cria um objeto a partir de uma classe, está dizendo ao computador para reservar um espaço na memória e dar vida ao que foi definido na classe. Segue um exemplo de um objeto abaixo

```
Pessoa p = new Pessoa(); // Criando um objeto da classe Pessoa  
p.nome = "Maria"; // Atribuindo valores aos atributos do objeto  
p.idade = 30;  
p.Falar(); // Saída: Olá, eu sou Maria
```



No exemplo acima, p é o objeto que foi criado a partir da classe Pessoa. Ele tem o nome "Maria" e a idade 30, além de usar o método falar.



ENCAPSULAMENTO

O encapsulamento é o conceito de esconder os detalhes internos de um objeto e fornecer uma interface controlada para interagir com ele. Isso significa que você pode limitar o acesso direto a certas partes de um objeto, mantendo dados importantes protegidos e acessíveis apenas por meio de métodos definidos.

Como funciona em C#?

Através de modificadores de acesso como `private`, `public` e `protected`.

private: O membro só pode ser acessado dentro da própria classe.

public: O membro pode ser acessado de qualquer lugar.

protected: O membro pode ser acessado dentro da classe e suas subclasses.

```
class Pessoa { private string nome; // Atributo encapsulado

    public void SetNome(string nome) { this.nome = nome; // Método para alterar o nome }

    public string GetNome() { return nome; // Método para acessar o nome }

}
```



POR QUE USAR ENCAPSULAMENTO?

Página 06

01

Segurança

Proteger os dados sensíveis de um objeto.

02

Manutenção

Facilita a atualização e a correção de erros, pois os detalhes internos podem mudar sem afetar o restante do código.

03

Controle

Permitir que apenas métodos específicos alterem os dados, garantindo que eles sejam sempre válidos.



HERANÇA

Herança é o mecanismo que permite que uma classe herde os atributos e métodos de outra classe. A herança promove a reutilização de código, pois evita que precisemos escrever código repetido.



```
class Animal {  
    public void Comer() {  
        Console.WriteLine("Animal está  
        comendo.");  
    }  
}  
  
class Cachorro : Animal {  
    public void Latir() {  
        Console.WriteLine("Cachorro está  
        latindo.");  
    }  
}
```



HERANÇA NA PRÁTICA

Explicação: No exemplo anterior, a classe Cachorro herda da classe Animal, o que significa que ela tem acesso ao método Comer(). Além disso, ela pode definir seus próprios métodos, como Latir().

```
Cachorro c = new Cachorro();
c.Comer(); // Saída: Animal está comendo.
c.Latir(); // Saída: Cachorro está latindo.
```



POLIMORFISMO

Polimorfismo permite que um objeto de uma classe derivada seja tratado como um objeto da classe base, enquanto ainda mantém comportamentos próprios. Isso é útil quando queremos que diferentes objetos executem a mesma ação de maneiras diferentes.

```
class Animal {  
    public virtual void FazerSom() {  
        Console.WriteLine("Animal faz som.");  
    }  
}  
  
class Gato : Animal {  
    public override void FazerSom() {  
        Console.WriteLine("Gato mia.");  
    }  
}
```



POLIMORFISMO NA PRÁTICA

O método FazerSom() pode ser sobreescrito em subclasses, permitindo comportamentos diferentes para diferentes tipos de animais.

```
Animal a = new Gato();  
a.FazerSom(); // Saída: Gato mia.
```

SOBRECARGA DE MÉTODOS

A sobrecarga de métodos permite que uma classe tenha vários métodos com o mesmo nome, desde que eles tenham assinaturas diferentes (diferentes parâmetros).

```
class Calculadora {  
    public int Somar(int a, int b) {  
        return a + b;  
    }  
  
    public double Somar(double a, double b) {  
        return a + b;  
    }  
}
```



BENEFÍCIOS DA PROGRAMAÇÃO ORIENTADA A OBJETOS

Reutilização de Código: Com a herança, podemos reutilizar código entre classes.

Manutenção: O encapsulamento permite que as alterações sejam feitas de maneira mais controlada e segura.

Expansibilidade: O polimorfismo facilita a adição de novos comportamentos sem alterar o código existente.



DESVANTAGENS DA POO

Complexidade: Pode ser difícil para iniciantes entender todos os conceitos da POO.

Custo de desempenho: Programas orientados a objetos podem ser menos eficientes em alguns casos, pois a criação de objetos consome mais recursos.





CONCLUSÃO

Página 14

A Programação Orientada a Objetos (POO) é um dos pilares mais importantes do desenvolvimento moderno de software. Ao introduzir conceitos como encapsulamento, herança e polimorfismo, ela nos permite criar sistemas mais organizados, flexíveis e reutilizáveis. Esses princípios não apenas facilitam o processo de desenvolvimento, mas também tornam a manutenção e a expansão dos projetos muito mais simples e eficientes.

No contexto de C#, a POO é essencial para a criação de aplicativos robustos, onde cada parte do código é modular, podendo ser alterada ou aprimorada sem comprometer o sistema como um todo. Ao utilizar classes e objetos, conseguimos modelar o mundo real no software, criando soluções que são intuitivas e escaláveis.

Embora o conceito possa parecer complexo no início, com a prática, a POO se torna uma ferramenta poderosa para resolver problemas complexos de maneira elegante. Com o uso adequado dos princípios da POO, como os apresentados nesta cartilha, você estará preparado para enfrentar desafios maiores e construir sistemas de software mais eficientes e bem organizados.

BIBLIOGRAFIA

C# Programming Guide - Microsoft Docs. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/csharp/>

Herbert Schildt - C# Completo e Total: do Básico ao Avançado. São Paulo: McGraw-Hill, 2017.

Andrew Troelsen - Pro C# 8.0 and the .NET Core 3.0: Explore the .NET Core 3.0 framework using C# 8.0. Apress, 2019.

Robert C. Martin - Código Limpo: Habilidades Práticas do Agile Software. São Paulo: Alta Books, 2016.

S. Dasgupta, C. Papadimitriou, U. Vazirani - Algoritmos. São Paulo: Cengage Learning, 2008.

Alura Cursos Online - Programação Orientada a Objetos com C# e .NET. Disponível em:
<https://www.alura.com.br>