



Nivelamento de Lógica de Programação e OO – Aula 3

Samira Antunes

Tópicos de hoje:

- Paradigma
 - POO
 - Diagrama de classe
 - Classe
 - Tipos de Atributos
 - Modificadores de Acesso
-

Combinados & Recados

Passaremos muito tempo juntos

- Câmera aberta, se possível.
- Levantar a mão em caso de dúvida.
- O representante da turma é o Ricardo Fazoli.
- Não esqueçam que teremos a rubrica de autoavaliação, avaliação da instrutora e avaliação do curso.
- A Caixa terá acesso dessa avaliação.
- Reposição de aula.
- Samira validar com a ADA se é possível fazer no dia 23/09 duas aula em uma.



Paradigmas

Mundo “real”

E, então?

- Como meu código pode ser semelhante ao mundo real?

*Em C#, usamos **classes** para descrever objetos do mundo real (carros, contas bancárias, livros).*

*Cada objeto tem **atributos** (dados) e **métodos** (ações) que correspondem às características e comportamentos que vemos na vida real.*

Mundo “real”

E, então?

- A classe Pessoa modela características (Nome, Idade) e comportamentos (Falar) assim como uma pessoa de verdade.
- Quando instanciamos `new Pessoa()`, temos um objeto que se comporta como o “mundo real” que ele representa.

Paradigmas de Programação

Visão geral

- Um paradigma é o “estilo” ou “modelo mental” de como organizamos e escrevemos código.
- No C# usamos o paradigma orientado a objetos como base.
- Criamos classes que descrevem o que o objeto é (atributos) e o que ele faz (métodos). Isso facilita entender e modelar problemas do mundo real em código.

Paradigmas de Programação

Imperativo

- Descreve como o computador deve executar as tarefas, passo a passo.
- Baseado em comandos, variáveis e controle de fluxo (if, for, while).

```
int soma = 0;
for (int i = 0; i < 10; i++)
    soma += i;
```


Paradigmas de Programação

Funcional

- Baseado em funções puras (sem efeitos colaterais) e imutabilidade.
- Foco em o que precisa ser feito, não em como.
- Em C#, vemos traços com LINQ e expressões lambda.
 - LINQ (Language Integrated Query) é um recurso do C# que permite escrever consultas de dados de forma parecida com SQL, mas dentro do próprio código C#.
 - Expressões lambda são funções anônimas (sem nome) que podem ser passadas como parâmetro.

Paradigmas de Programação

Funcional

- Esses dois recursos aproximam o C# do paradigma funcional, porque incentivam:
 - Imutabilidade (não mudar variáveis originais).
 - Funções puras (mesma entrada → mesma saída).

```
var numeros = new[] {1,2,3,4};  
var pares = numeros.Where(n => n % 2 == 0);
```

- Where recebe uma lambda ($n \Rightarrow n \% 2 == 0$) que retorna apenas os pares.
- Você não alterou o array original, apenas filtrou → estilo funcional.

Paradigmas de Programação

Orientado a Objetos (OOP / POO)

- Modelo que se aproxima de como enxergamos o mundo real: objetos com propriedades (dados/atributos) e comportamentos (métodos).
- C# é fortemente orientado a objetos.

```
public class Pessoa
{
    public string Nome { get; set; }
    public void Falar() => Console.WriteLine($"Olá, sou {Nome}");
}
```

Programação Orientada a Objetos (POO)

O que é Programação Orientada a Objetos (POO)

Entendimento

- POO é um jeito de modelar o problema pensando em objetos, que são representações de coisas do mundo real ou de conceitos.
- Cada objeto tem:
 - Propriedades/Atributos: características (dados).
 - Métodos/Comportamentos: ações que ele sabe fazer.
- Classe: é o molde (plano de construção).
- Objeto: é a instância real criada a partir desse molde.

O que é Programação Orientada a Objetos (POO)

Entendimento

- Vantagens:
 - Organização: o código fica mais próximo do mundo real.
 - Reutilização: você pode criar várias instâncias sem reescrever lógica.
 - Encapsulamento: protege detalhes internos (futuro assunto da aula 4).
 - Facilidade de manutenção: cada objeto cuida da sua própria lógica.

O que é Programação Orientada a Objetos (POO)

Entendimento

- Pense que a POO é como montar um jogo de Lego.
- Cada pecinha é um objeto com suas características e funções.
- Você projeta a peça (classe) e depois cria quantas quiser (objetos), encaixando tudo para formar um sistema completo.

O que é Programação Orientada a Objetos (POO)

Entendimento

```
public class Pessoa
{
    public string Nome { get; set; } // propriedade
    public int Idade { get; set; }   // propriedade

    public void Falar()              // método
    {
        Console.WriteLine($"Olá, meu nome é {Nome}");
    }
}
```

```
Pessoa p = new Pessoa();
p.Nome = "Ana";
p.Idade = 25;
p.Falar(); // Saída: Olá, meu nome é Samira
```

- Pessoa é a classe (molde).
- p é o objeto (instância real).
- Nome e Idade são as propriedades.
- Falar() é o comportamento.



Diagrama de Classe

Diagrama de Classe

Conceito

- UML (Unified Modeling Language) é uma linguagem visual para representar sistemas.
- O Diagrama de Classe descreve as classes do sistema, seus atributos (dados) e métodos (funções).

Diagrama de Classe

Estrutura básica

- Uma classe em UML é representada por um retângulo dividido em 3 partes:
 - Nome da Classe.
 - Atributos (propriedades/dados).
 - Métodos (comportamentos).
- Notação de Visibilidade
 - + → public (pode ser acessado de fora da classe).
 - - → private (acessível apenas dentro da própria classe).
 - # → protected (acessível na própria classe e nas classes filhas).

Diagrama de Classe

Estrutura básica

```
+-----+
|      Pessoa      |
+-----+
| - nome : string  |
| - idade : int    |
+-----+
| + Falar() : void  |
| + Aniversario() : void |
+-----+
```

```
public class Pessoa
{
    private string nome;
    private int idade;

    public void Falar()
    {
        Console.WriteLine($"Olá, meu nome é {nome}");
    }

    public void Aniversario()
    {
        idade++;
    }
}
```

Diagrama de Classe

Resumo

- O diagrama de classes é como o blueprint (planta) da nossa classe em C#.
- O sinal + quer dizer que é público, qualquer um pode acessar.
- O - indica que é privado, apenas a própria classe pode mexer.
- Assim, entendemos a estrutura antes de escrever o código.



Criando classe na prática

Classe

Visão geral

- Classe é como um molde ou planta para criar objetos.
- Ela descreve quais dados (atributos) e quais ações (métodos) um objeto vai ter.

Classe

Visão geral

```
public class Pessoa
{
    public string Nome;
    public int Idade;

    public void Falar()
    {
        Console.WriteLine($"Olá,
meu nome é {Nome}");
    }
}
```

- public class Pessoa
 - public: a classe pode ser usada em qualquer lugar do programa.
 - Pessoa: é o nome do molde.
- public string Nome; e public int Idade;
 - Nome e Idade são atributos (informações que cada pessoa terá).
 - string é texto; int é número inteiro.
- public void Falar()
 - Falar é um método (ação/comportamento).
 - Ele escreve no console: "Olá, meu nome é ...".

Classe

Visão geral

```
public class Pessoa
{
    public string Nome;
    public int Idade;

    public void Falar()
    {
        Console.WriteLine($"Olá,
meu nome é {Nome}");
    }
}
```

- Em C# qualquer membro de uma classe (variáveis, propriedades ou métodos) pode ter um modificador de acesso como public ou private.
- O que muda é quem consegue “enxergar” e usar esse membro.
- Nome é public → qualquer parte do código pode fazer.
- private → só a própria classe Pessoa consegue usar.

Classe

Visão geral

```
public class Pessoa
{
    public string Nome;
    public int Idade;

    public void Falar()
    {
        Console.WriteLine($"Olá,
meu nome é {Nome}");
    }
}
```

- Proteção dos dados: impede alterações diretas.
- Encapsulamento: você controla o acesso criando métodos ou propriedades seguras.
- public: “porta aberta”, qualquer classe pode acessar.
- private: “porta fechada”, apenas a própria classe acessa.
- Então, sim, variáveis (campos) podem ser públicas ou privadas, mas a boa prática é deixá-las private e expor apenas o necessário por meio de propriedades ou métodos.

Criando um objeto (instanciando)

Visão geral

```
Pessoa p = new Pessoa();
```

- Para usar a classe, criamos um objeto.
- Pessoa p: declara a variável p do tipo Pessoa.
- new Pessoa(): cria uma nova instância (um objeto real baseado no molde).
- Agora p é “uma pessoa” que existe na memória do programa.

Atribuindo valores

Visão geral

```
p.Nome = "Samira";  
p.Idade = 37;
```

- Preenchemos os atributos desse objeto: nome "Samira" e idade 37.

Chamando o método

Visão geral

```
p.Falar();
```

- Executa a ação Falar().
- O console mostrará: Olá, meu nome é Samira

Resumo

Visão geral

- Classe Pessoa → Molde de um “boneco” (define que todo boneco tem Nome, Idade e sabe Falar).
- `new Pessoa()` → Faz um boneco novo.
- `p.Nome = "Ana"` → Escreve “Ana” na etiqueta desse boneco.
- `p.Falar()` → O boneco “fala” usando os dados que você colocou.

Tipos de Atributos

Tipos de Atributos

Visão geral

- Em C#, um atributo é um dado associado a uma classe ou objeto. Eles podem ser representados de duas formas.
- Campos (Fields):
 - Variáveis declaradas diretamente dentro da classe.
 - Podem ser públicas ou privadas.
 - Exemplo:

```
public string nome; // Campo
```

Tipos de Atributos

Visão geral

- Em C#, um atributo é um dado associado a uma classe ou objeto. Eles podem ser representados de duas formas.
- Propriedades (Properties):
 - Variáveis declaradas diretamente dentro da classe.
 - Podem ser públicas ou privadas.
 - Exemplo:

```
public string Nome { get; set; } // Propriedade simples
public int Idade { get; private set; } // Somente leitura externa
```

Diferença Prática

Visão geral

- Campo: dado cru, acesso direto.
- Propriedade: acesso controlado (pode validar, calcular, etc.).

Atributos de Instância x Atributos Estáticos

Tipo	Como funciona	Exemplo
Instância	Cada objeto tem sua própria cópia.	<code>p1.Nome</code> diferente de <code>p2.Nome</code>
Estático (static)	Pertence à classe, compartilhado por todos os objetos.	<code>Pessoa.Contador</code> é único

Diferença Prática

Visão geral

```
public class Pessoa
{
    public string Nome;           // Instância
    public static int TotalPessoas; // Estático
}
```

- TotalPessoas é o mesmo valor para todos os objetos.

Modificadores de Acesso

Modificadores de Acesso

Visão geral

- Controlam quem pode ver ou usar um membro (atributo, método, classe). Pense em “portas abertas ou fechadas”:

Modificador	Quem pode acessar	Analogia
public	Qualquer parte do projeto	Porta totalmente aberta
private	Apenas dentro da própria classe	Porta trancada
protected	Classe atual + classes que herdam	Porta aberta só para a “família”
internal	Qualquer classe no mesmo projeto/assembly	Porta aberta só para quem mora no mesmo prédio

Modificadores de Acesso

Visão geral

```
public class Pessoa
{
    public string Nome;           // Todos acessam
    private int idade;           // Só a própria classe
    protected string Documento;  // Classe e herdeiros
    internal string Registro;     // Dentro do mesmo assembly
}
```

Prática

Lista de exercícios:

1. Paradigmas/P00

- Explique em suas palavras o que é P00 e dê um exemplo do mundo real que se encaixe como uma classe.
- É opcional a adição do diagrama UML.

2. Classe e Objeto

- Crie a classe Carro com atributos Marca, Ano e um método Ligar() que exiba “Carro ligado!”.
- É opcional a adição do diagrama UML.

3. Modificadores

- Altere a classe Carro para que o atributo Ano seja private e crie um método público que retorne esse valor.
- É opcional a adição do diagrama UML.

Exercícios Práticos

Integração dos conceitos

Lista de exercícios:

Entrega: até 21/09/2025.

Envio para o e-mail da professora.

Obrigada