

Nivelamento de Lógica de Programação e OO – Aula 2

Samira Antunes

# Tópicos de hoje:

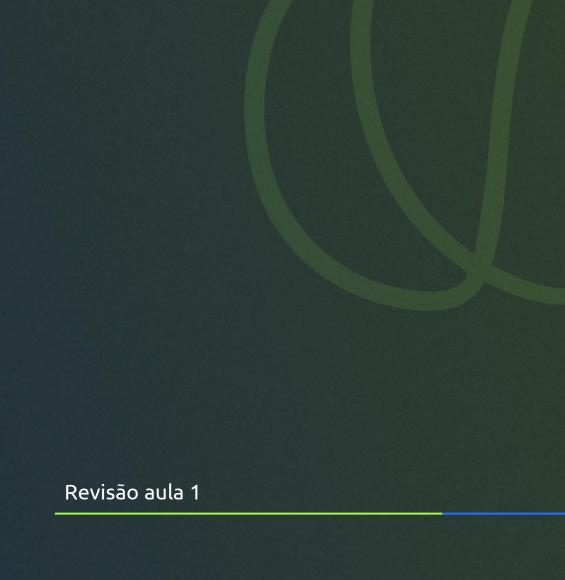
- Revisão aula 1
- Métodos
- Estruturas de repetição
- Dicionários
- Tratamento de erros
- Prática

### 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.
- O projeto final será compartilhado na próxima aula.
- Não esqueçam que teremos a rubrica de autoavaliação, avaliação da instrutora e avaliação do curso.
- A Caixa terá acesso dessa avaliação.





• Quem lembra o que é interpolação de strings?

É uma forma prática de inserir variáveis dentro de um texto sem precisar usar concatenação com + Basta usar o \$ antes da string e {colocar a variável entre chaves}.



```
string nome = "Samira";
int idade = 36;

Console.WriteLine($"Olá, {nome}! Você tem {idade} anos.");
// Saída: Olá, Samira! Você tem 36 anos.

Console.WriteLine("Olá, " + nome + "! Você tem " + idade + " anos.");
// Saída: Olá, Samira! Você tem 36 anos.
```

"Interpolação de strings: usar \$"texto {
{variável}" para inserir valores em uma string de forma simples e legível."



- Instalação e configuração do Visual Studio / .NET
- Estrutura de uma aplicação C# Console
- Entrada e saída de dados com Console.Write e Console.ReadLine
- Tipos primitivos: int, double, bool, char, string
- Constantes e variáveis, inferência com var
- Operadores aritméticos (+, -, \*, /, %)



- Conversão de tipos (int.Parse, double.Parse, ToString)
- Interpolação e concatenação de strings
- Estruturas if / else e switch
- Operadores ternário (? :) e coalescência (??, ?.)
- Prática Realizada:
  - Vários exemplos de código executados no Visual Studio.
  - Exercícios de fixação: calculadora, conversor de temperatura, classificador de números, cadastro de aluno.



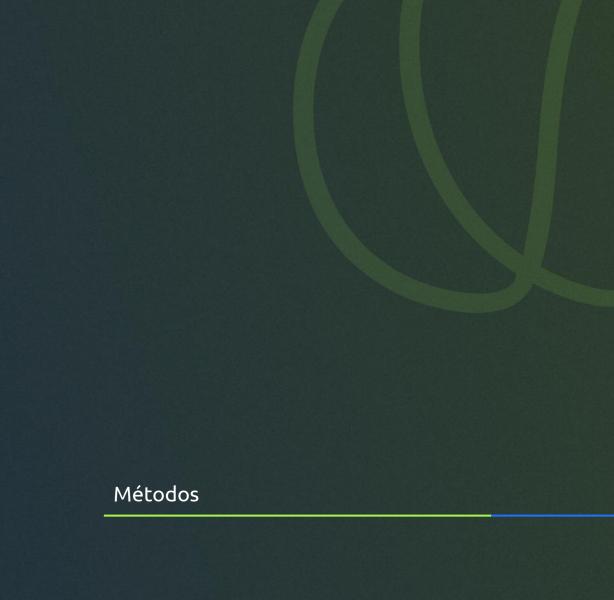


# Vale atenção

Próximos Passos

- Aula 3 veremos:
  - Na Aula 3 veremos a introdução oficial à Programação Orientada a Objetos (POO) — classes, objetos, atributos, modificadores de acesso e representação em UML.
- Aula 4 veremos:
  - Aprofundamento em construtores, encapsulamento e propriedades.
  - As palavras-chave (namespace, public, class, etc.) fazem parte da organização do código em Programação Orientada a Objetos.





### Entendimento dos Métodos

- São blocos de código que executam uma tarefa e podem receber parâmetros e retornar valores.
  - Assinatura = nome + parâmetros + tipo de retorno.

```
// Método simples
public static void Saudacao()
{
    Console.WriteLine("Bem-vindos!");
}

// Método com retorno e parâmetros
public static int Somar(int a, int b)
{
    return a + b;
}
```



### Modificadores de Acesso

Visibilidade

- Controlam a visibilidade dos membros:
  - public visível em qualquer parte do projeto.
  - private visível apenas na própria classe.
  - internal visível apenas no mesmo assembly.
  - protected visível na classe e em subclasses.

```
public class Calculadora
{
    private int contador = 0;
    public int Somar(int a, int b) => a + b;
}
```



Estruturas de repetição

# A repetição

- As estruturas de repetição permitem executar um bloco de código várias vezes, de acordo com uma condição ou até que um conjunto de dados seja percorrido.
- São essenciais para automatizar tarefas e evitar código repetitivo.
- while verifica a condição antes, do…while garante pelo menos uma execução, for é ideal para contagens e foreach percorre coleções.



- Verifica a condição antes de cada iteração.
- Executa enquanto a condição for verdadeira.
- Se a condição inicial for falsa, o bloco não executa nenhuma vez.

```
int i = 0;
while (i < 5)
{
     Console.WriteLine(i);
     i++;
}</pre>
```



- Executa o bloco pelo menos uma vez, depois verifica a condição.
- Útil quando você precisa que o código rode pelo menos uma vez (ex.: menu interativo).

```
int j = 0;
do
{
     Console.WriteLine(j);
     j++;
} while (j < 5);</pre>
```



- Ideal quando sabemos o número de repetições.
- Possui 3 partes: inicialização, condição e incremento/decremento.

```
for (int k = 0; k < 5; k++)
{
    Console.WriteLine(k);
}</pre>
```

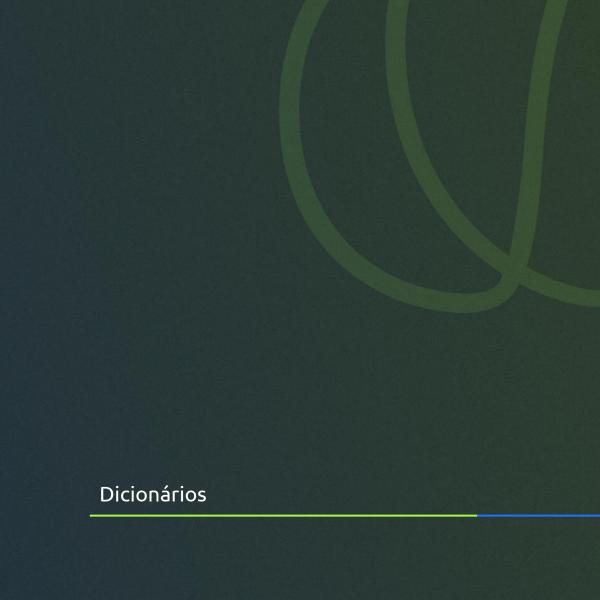


foreach

• Usado para listas, arrays, dicionários e outras coleções.

```
string[] frutas = { "Maçã", "Banana", "Uva" };
foreach (string f in frutas)
{
    Console.WriteLine(f);
}
```





- Coleção de pares chave-valor (Key-Value).
- Permite armazenar e recuperar informações pelo nome da chave, e não pelo índice numérico como em arrays ou listas.
- Chave é única dentro do dicionário (não pode repetir).
- O valor pode ser de qualquer tipo e pode se repetir.
- Acesso rápido: busca e inserção são muito eficientes.



- Operações Comuns:
  - Add(chave, valor) → adiciona um par.
  - Remove(chave) → remove a chave e seu valor.
  - ContainsKey(chave) → verifica se a chave existe.
  - TryGetValue(chave, out valor) → obtém o valor de forma segura.
  - Clear() → limpa todo o dicionário.
- Quando usar:
  - Quando você precisa de acesso rápido a um valor conhecido por uma chave única.
  - Ex.: Estoque de produtos, cadastro de alunos (matrícula → nome), configuração de parâmetros.



- Dictionary<TKey, TValue> é uma coleção de pares chavevalor com busca rápida por chave única.
- Ideal para armazenar e consultar dados identificados por um nome/código em vez de índice.



```
using System;
using System.Collections.Generic;
Dictionary<string, int> estoque = new Dictionary<string, int>();
// Adicionando pares
estoque.Add("Maçã", 10);
estoque["Banana"] = 5; // outra forma de adicionar
// Acessando valores
Console.WriteLine(estoque["Maçã"]); // imprime 10
// Atualizando
estoque["Maçã"] = 12;
// Verificando se a chave existe
if (estoque.ContainsKey("Banana"))
    Console.WriteLine("Tem banana no estoque!");
```



# Tratamento de Erros

### Tratamento de erros

- Por que tratar erros?
  - Durante a execução, seu programa pode enfrentar exceções: divisão por zero, arquivo inexistente, entrada inválida etc.
  - Sem tratamento, o programa encerra abruptamente.
  - Com tratamento, você previne falhas, informa o usuário e mantém o controle.



### Tratamento de erros

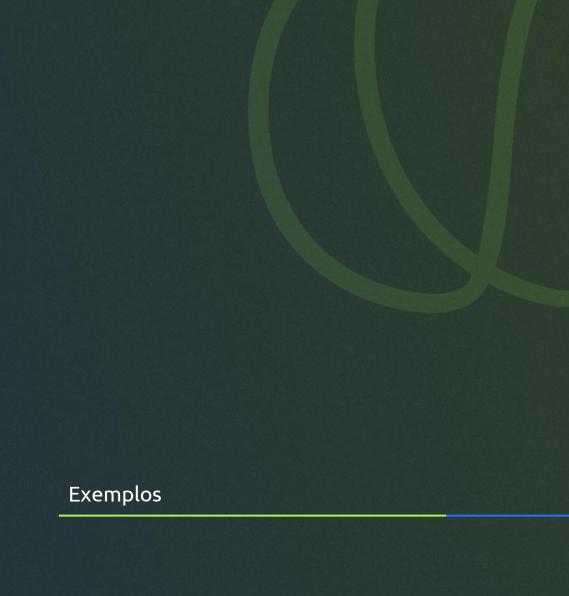
```
try
    Console.Write("Digite um número: ");
    int n = int.Parse(Console.ReadLine());
    Console.WriteLine(10 / n);
catch (FormatException ex)
    Console.WriteLine("Entrada inválida: " + ex.Message);
catch (DivideByZeroException)
    Console.WriteLine("Não é possível dividir por zero!");
finally
    Console.WriteLine("Fim do processo.");
```



### Tratamento de erros

- Se o usuário digitar algo que não é número → FormatException.
- Se digitar 0 → DivideByZeroException.
- finally roda sempre, mesmo se houver erro.
- Use try para o código que pode falhar, catch para tratar a falha, finally para limpar recursos e throw para lançar erros. Isso mantém o programa seguro e evita encerramentos inesperados.





### Menu no Console

Aplicação

```
int opcao;
do
{
    Console.WriteLine("1 - Somar\n2 - Subtrair\n0 - Sair");
    opcao = int.Parse(Console.ReadLine());

    if (opcao == 1) Console.WriteLine("Soma!");
    else if (opcao == 2) Console.WriteLine("Subtração!");
} while (opcao != 0);
```



### Loteria – 6 números aleatórios

Aplicação

```
Random rnd = new Random();
for (int i = 0; i < 6; i++)
{
   int numero = rnd.Next(1, 61);
   Console.WriteLine(numero);
}</pre>
```





Integração dos conceitos

# Lista de exercícios:

# 1. Métodos

- Crie um método double Media(double a, double b, double c) que receba três notas, calcule e retorne a média.
- Em seguida, dentro da mesma classe, chame o método para exibir o resultado.

# 2. Estruturas de Repetição

- Implemente um programa que leia números inteiros até o usuário digitar 0.
- Use um while para somar todos os números digitados (exceto o 0) e exiba a soma final.



Integração dos conceitos

# Lista de exercícios:

- 3. Dicionários
  - Crie um Dictionary<string,int> para armazenar o nome de três produtos e suas quantidades em estoque.
  - Peça ao usuário nome e quantidade de cada produto.
  - Depois, percorra o dicionário com foreach e mostre:
     Produto: X Quantidade: Y.



Integração dos conceitos

# Lista de exercícios:

- 4. Tratamento de Erros
  - Solicite ao usuário dois números inteiros e divida o primeiro pelo segundo.
  - Use try/catch para:
    - 🗸 Detectar entrada inválida (FormatException).
    - ✓ Tratar divisão por zero (DivideByZeroException).
    - ✓ No finally, exibir "Fim do cálculo".



Integração dos conceitos

# Lista de exercícios:

Entrega: até 21/09.

Envio do projeto para o e-mail da professora.



Obrigada