



Fundamentos de Desenvolvimento com C#

Aula 02: Depuração e Strings

Professor: Rinaldo Ferreira Junior

E-mail: rinaldo.fjunior@prof.infnet.edu.br



- **Professor:** Rinaldo Ferreira Junior
- **Graduação:** Pós-graduado em Arquitetura de Softwares
- **Atuação:** .Net | C# | SQL | NoSQL | Engenheiro de Software
- **E-mail:** rinaldo.fjunior@prof.infnet.edu.br
- **Linkedin:** <https://www.linkedin.com/in/rinaldo-ferreira-junior-787326a>

- Tipagem
- Value e Reference Types
- Variáveis e escopo
- Strings
- Operadores Aritméticos
- Depuração

- Strongly Typed: Uma linguagem possui tipagem forte, quando é obrigatória a especificação dos tipos de dados de variáveis e objetos.
- Weakly Typed: Uma linguagem possui tipagem fraca, quando não é obrigatória a especificação dos tipos de dados de variáveis e objetos.
- C#: É uma linguagem de tipagem forte.
- Na tipagem forte, o compilador verifica se os tipos usados no Código atendem ao conjunto de regras estabelecidas para cada tipo de dado.

- Value Types
 - Contém diretamente uma instância daquele valor.
 - Cada variável possui sua cópia do dado.
 - Operações em uma variável, não afetam outras.
- Reference Types
 - Armazenam uma referência para o dado real.
 - Duas variáveis podem referenciar o mesmo dado.
 - Operações em uma variável podem afetar a outra.

Tipo C#	Tipo .Net
<u>bool</u>	<u>System.Boolean</u>
<u>byte</u>	<u>System.Byte</u>
<u>sbyte</u>	<u>System.SByte</u>
<u>char</u>	<u>System.Char</u>
<u>decimal</u>	<u>System.Decimal</u>
<u>double</u>	<u>System.Double</u>
<u>float</u>	<u>System.Single</u>
<u>int</u>	<u>System.Int32</u>
<u>uint</u>	<u>System.UInt32</u>
<u>nint</u>	<u>System.IntPtr</u>
<u>nuint</u>	<u>System.UIntPtr</u>
<u>long</u>	<u>System.Int64</u>
<u>ulong</u>	<u>System.UInt64</u>
<u>short</u>	<u>System.Int16</u>
<u>ushort</u>	<u>System.UInt16</u>

Tipo C#	Tipo .Net
<u>object</u>	<u>System.Object</u>
<u>string</u>	<u>System.String</u>
<u>dynamic</u>	<u>System.Object</u>

- Uma variável é uma identificação dada à uma região de memória, que armazena algum valor.
- Variáveis precisam de um nome e um tipo de dados e podem, opcionalmente, ser inicializadas com algum valor.
- Algumas regras devem ser seguidas, para a criação de variáveis:
 - O nome deve ser único, iniciando com uma letra, e devem conter apenas letras, dígitos e o caracter _.
 - Os nomes são case-sensitive, portanto, name e Name são nomes diferentes.
 - Palavras reservadas, como int ou string, não podem ser usadas como nomes (é uma restrição do compilador) à menos que se coloque um símbolo @ na frente do nome

- Variáveis podem ser declaradas explicitamente:

```
static void Main(string[] args)
{
    string phrase = "Hello, world";
    int version = 1;

    Console.WriteLine(phrase + " versão " + version);
}
```

- Ou implicitamente

```
static void Main(string[] args)
{
    var phrase = "Hello, world";
    var version = 1;

    Console.WriteLine(phrase + " versão " + version);
}
```

 (local variable) int version

- Variáveis possuem escopos, ou seja, áreas onde podem ser acessadas.

```
internal class Program
{
    static int limit = 10;  —————> Escopo de classe

    0 references
    static void Main(string[] args)
    {
        string sentence = "Iteração: ";  —————> Escopo local

        for (int i = 0; i < limit; i++) —————> Escopo de bloco
        {
            Console.WriteLine(sentence + i + " de: " + limit);
        }

        Console.Read();
    }
}
```

- Strings são usadas para armazenar texto. Internamente, esse texto é armazenado como uma coleção read-only de chars.
- Cada char ocupa 2 bytes de memória
- Strings são imutáveis, ou seja, não podem ser alteradas após terem sido criadas.
- Todos os operadores e métodos que parecem alterar uma string, na verdade geram uma nova string com o resultado desejado.

- Se os operandos de uma expressão são do tipo string, o operador `+` concatena esses operandos, gerando uma nova string como resultado.
- A representação string de um nulo (null) é uma string vazia

```
string text = "Ops!" + "faltou um espaço"; // Ops!faltou um espaço  
string withNull = "Concatenando com nulo" + null; // Concatenando com nulo
```

- O caracter `$` identifica uma expressão string como uma string interpolada, que é um literal que possui expressões de interpolação.
- O compilador substitui os items da expressão, pelas suas representações em string.

```
internal class Program
{
    static int limit = 10;

    0 references
    static void Main(string[] args)
    {
        for (int i = 0; i < limit; i++)
        {
            Console.WriteLine($"Iteração {i} de: {limit}");
        }

        Console.Read();
    }
}
```

- São suportados por todos os tipos numéricos integrais e de ponto flutuante.

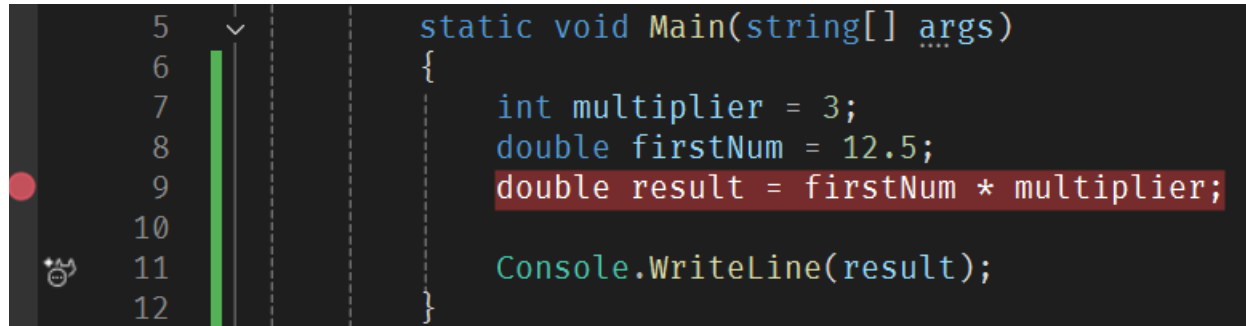
+ Adição
- Subtração
* Multiplicação
/ Divisão
% Resto

```
static void Main(string[] args)
{
    int multiplier = 3;
    double firstNum = 12.5;
    double result = firstNum * multiplier;

    Console.WriteLine(result);
}
```

- A depuração vincula automaticamente um depurador ao Código.
- Breakpoints pausam a execução do código no ponto onde são definidos. A partir daí, você pode inspecionar a execução do Código, linha por linha.
- F9 cria ou retira um breakpoint da linha onde estiver o cursor.
- F5 executa o código com o depurador vinculado.
- CTRL + F5 executa o código sem o depurador.
- F11 executa o código linha por linha
- F10 salta a execução de blocos de códigos externos

- O indicador e a linha vermelha, representam um breakpoint no local.



The screenshot shows a code editor with a dark background. On the left, a vertical line of numbers from 5 to 12 represents line numbers. A red dot, indicating a breakpoint, is positioned to the left of line 9. A vertical green line is positioned to the right of the red dot. The code on the right is as follows:

```
static void Main(string[] args)
{
    int multiplier = 3;
    double firstNum = 12.5;
    double result = firstNum * multiplier;
    Console.WriteLine(result);
}
```