C# COMPLETO Programação Orientada a Objetos + Projetos

Namespace: Agrupamento de classes relacionadas. É uma boa prática colocar as classes criadas dentro de namespaces;

# Primeiro Programa

Visual Studio Code com a extensão C# instalada;

* O SDK do .net 5,0;
* Criar a pasta em que seu nome será o nome do projeto;
* Abrir o VSCode
* E no terminal (posicionado na pasta criada) executar: ***dotnet new console --framework net5.0***
* O código de modelo define uma classe, Program , com um único método, Main.
* Main é o ponto de entrada do aplicativo, o método que é chamado automaticamente pelo runtime quando ele inicia o aplicativo. Quaisquer argumentos de linha de comando fornecidos quando o aplicativo for iniciado estão disponíveis na matriz args.

Texto

Descrição gerada automaticamente

Para executar o aplicativo: dotnet run

# Estrutura de um Program C#

.csproj: Arquivo de configuração

* .cs: Arquivo com o fonte em C#
* Pasta obj e bin: Arquivos compilados

# Lógica de Programação

Principais tipos de dados:

Tabela

Descrição gerada automaticamente

Declaração: int x = 10;

* Overflow: Cálculo extrapola o tamanho da variável fazendo com que a variável receba o valor do limite oposto (Exemplo1: byte n1 = 255 // n1++ // Resultado = 0; Exemplo2: sbyte n1 = 127 // n1++ // Resultado = -128)
* Ao utilizar long, acrescentar o “L” no final como boas práticas: long y = 246298356L
* Ao declarar um char, é possível utilizar o código Unicode (Exemplo: char letra = ’\u0041’)
* Char: Utilizar aspas simples; Strings: aspas duplas
* Variável float: utilizar um ‘f’ no final (Exemplo: float a = 1.75f)

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Toda classe em C# é uma subclasse de object, logo, ele pode receber tipos float, string, char e etc.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

# Algumas Convenções

Camel Case: lastName (parâmetros de métodos, variáveis dentro de métodos)

* Pascal Case: LastName (namespaces, classe, properties e métodos)
* Padrão \_lastName (atributos "internos" da classe)

# Combinar Strings

PlaceHolders: Console.WriteLine("{0} tem {1} anos e tem saldo igual a {2:F2} reais", nome, idade, saldo);

* Interpolação: Console.WriteLine($"{nome} tem {idade} anos e tem saldo igual a {saldo:F2} reais");
* Concatenação: Console.WriteLine(nome + " tem " + idade + " anos e tem saldo igual a " + saldo.ToString("F2", CultureInfo.InvariantCulture) + " reais"); Necessário: using System.Globalization;

# Operadores de Atribuição:

Tabela

Descrição gerada automaticamente

# Conversão de tipos

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/numeric-conversions>

* Conversão implícita: Quando a conversão de tipos de variáveis pode ocorrer naturalmente. Ex.: Converter de float -> double (de 4 para 8 bits);
* Casting (conversão explícita): b = (float)a (no caso o tipo de b é double); b = (int)a
* Se eu dividir um int por outro int, o compilador entende que eu quero apenas a parte inteira dessa divisão (mesmo que eu esteja atribuindo o valor p um double ou float) então é necessário utilizar o casting para receber o valor real dessa operação: int a; int b; double div = (double) a/b;

# Operadores Aritméticos

+, -, \*, /, %

# Entrada de Dados

Console.ReadLine() -> Stgring;

* Split: string[] vet = s.split(‘<Separador>’)
* Conversão de string pra outro tipo: <tipo>.Parse(Console.ReadLine()). Ex.: int.Parse(Console.ReadLine())

# Operadores Comparativos

Tabela

Descrição gerada automaticamente

# Operadores Lógicos

Tabela

Descrição gerada automaticamente

Precedência: ! > && > ||

# Estrutura Condicional

Interface gráfica do usuário, Aplicativo, Word

Descrição gerada automaticamente

# Funções

Sintaxe:

static <tipo retorno> Nome\_Funcao(<tipo> a, <tipo> b, ...) {

<comandos>

Return x

}

# Estrutura de repetição

While:

Word

Descrição gerada automaticamente com confiança baixa

For:

Linha do tempo

Descrição gerada automaticamente