**Linguagem de programação C#**

**O que é C# (C SHARP)?**

É uma linguagem de programação de alto nível.

**O que é uma linguagem de programação de alto nível?**

A linguagem de programação de alto nível está mais próxima do programador do que do dispositivo, se assemelhando mais com a linguagem humana. Ela tem o objetivo de facilitar a maneira de programar, pois é mais intuitiva e fácil de aprender.

**O que é uma linguagem de programação de baixo nível?**

A linguagem de programação de baixo nível é mais voltada ao dispositivo, e possui principal objetivo de se comunicar com o computador de maneira mais rápida, ao contrário da linguagem de alto nível.

**O que é linguagem de programação compilada?**

Linguagem compilada é uma linguagem de programação em que o código fonte, nessa linguagem, é executado diretamente pelo sistema operacional ou pelo processador, após ser traduzido por meio de um processo chamado compilação, usando um programa de computador chamado compilador, para uma linguagem de baixo nível, como linguagem de montagem ou código de máquina.

**O que é linguagem de programação interpretada?**

Linguagem interpretada é uma linguagem de programação em que o código fonte nessa linguagem é executado por um programa de computador chamado interpretador, que em seguida é executado pelo sistema operacional ou processador. Mesmo que um código em uma linguagem passe pelo processo de compilação, a linguagem pode ser considerada interpretada se o programa resultante não for executado diretamente pelo sistema operacional ou processador.

**O que é tipagem de dados?**

A linguagem de programação C# é uma linguagem do tipo tipada, ou seja, obrigam especificar o tipo de dado.

Exemplo:

INT idade = declarando uma variável chamada idade, do tipo numérico que aceita apenas números inteiros.

**TIPOS INTERNOS (REFERÊNCIA C#)**

|  |  |
| --- | --- |
| Palavra-chave Type do C# | Tipo .NET |
| bool | System.Boolean |
| byte | System.Byte |
| sbyte | System.SByte |
| char | System.Char |
| decimal | System.Decimal |
| double | System.Double |
| float | System.Single |
| int | System.Int32 |
| uint | System.UInt32 |
| nint | System.IntPtr |
| nuint | System.UIntPtr |
| long | System.Int64 |
| ulong | System.UInt64 |
| short | System.Int16 |
| ushort | System.UInt16 |

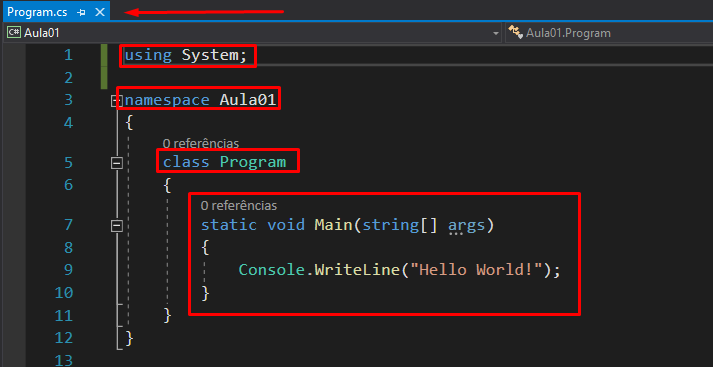
**Porque utilizar C#?**

Pois, é uma linguagem programação completa, além de ser utilizada para desenvolvimento web e mobile pode ser utilizada para desenvolvimento de games, entre outros. Além disso, é uma tecnologia que está no mercado a um bom tempo, isso pode ser uma boa vantagem, pois consequentemente tem um grande mercado de trabalho e muitos materiais para estudo na internet.

**Como o C# funciona?**

Ela é uma linguagem de programação tipada, compilada e de código gerenciado, sua execução depende de um gerenciador chamado “Runtime” ele é responsável por limpar, alocar a memória e como alocar essas informações na memória, desta maneira ficamos mais focados no desenvolvimento da linguagem do que nos recursos de máquina.

**Escopo de um programa em C#**



**Definições**

* Importações
* Namespace
* Classe
* Método principal

**O que é um namespace?**

Namespace são divisões lógicas, não podemos ter duas classes com o mesmo nome em um único namespace, também pode ser utilizado em diversos arquivos.

**O que é Using?**

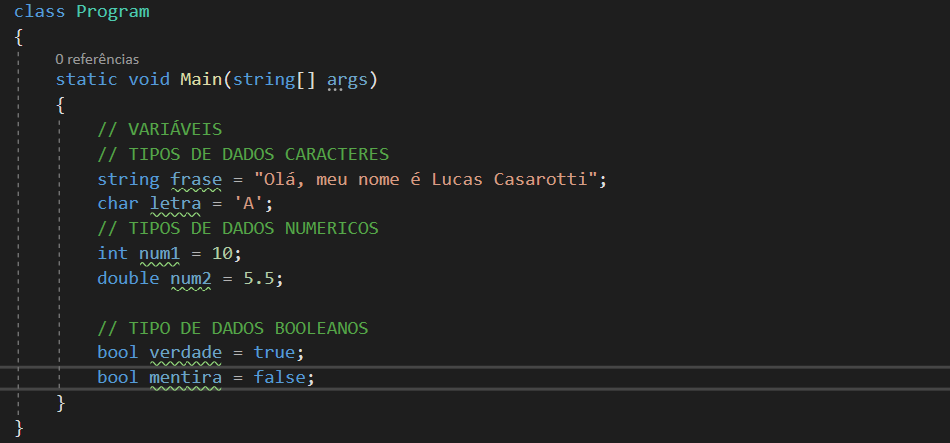
O Using define qual bibliotecas iremos utilizar no arquivo, outros namespace podem ser importados através do Using sendo eles internos ou externos, por padrão só vem incluso o básico, outras bibliotecas precisam ser importadas.

**Resumindo a execução do escopo**

Busca o program.cs, lê a classe program e lê o método main(principal).

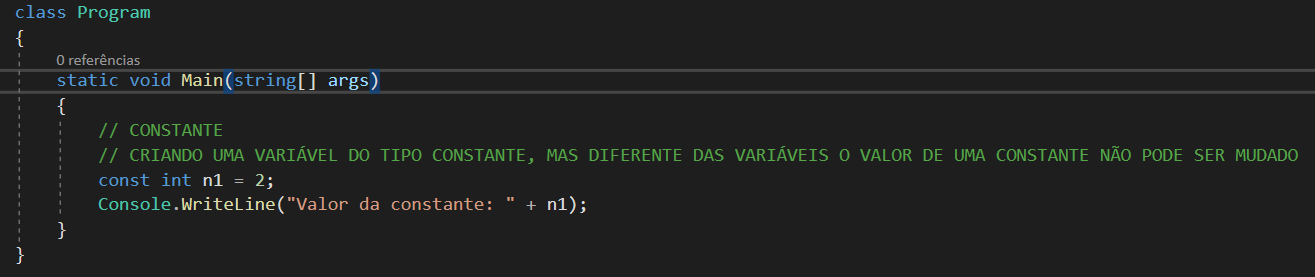
**Variáveis**

Variáveis são espaços na memória utilizados para armazenar algo, como seu próprio nome diz ela é variável, ou seja, seu valor pode ser alterado a qualquer momento. Sempre que criarmos uma varável, dizemos que estamos inicializando uma variável.



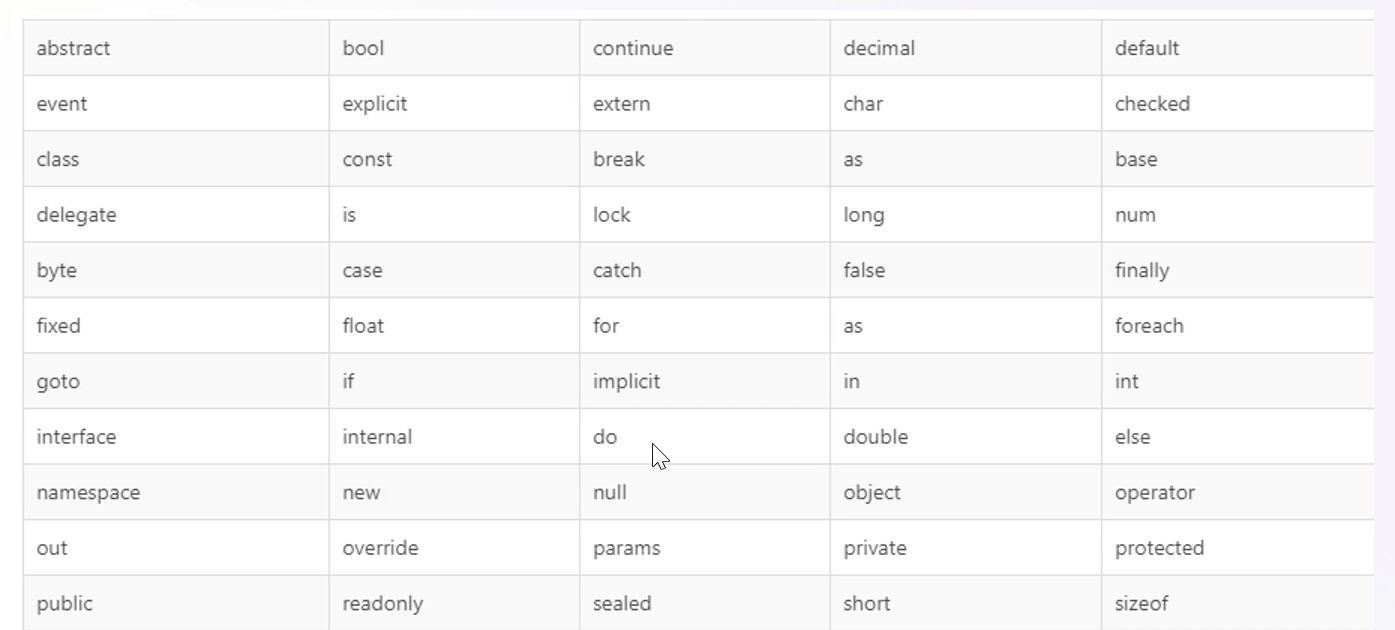
**Constantes**

Também utilizamos as constantes para armazenar valores, mas diferente das variáveis o valor atribuído a uma constante não pode ser modificado.



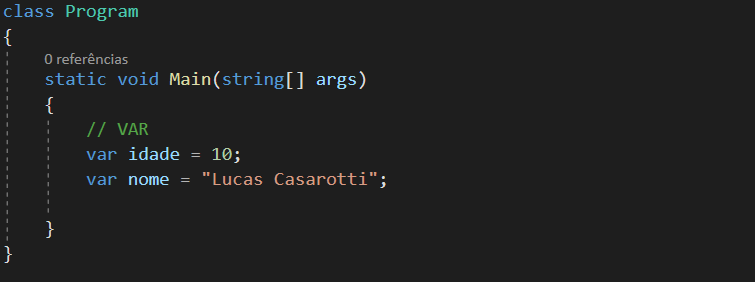
**Nomes reservados**

Também chamados de Keywords, são palavras exclusiva do C# que não pode ser utilizadas como nomes de variáveis.

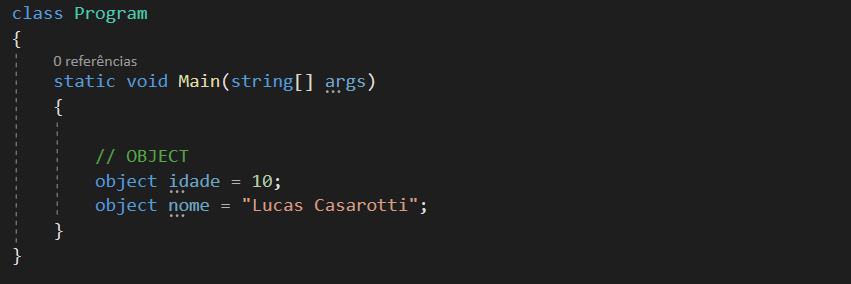


**Var e Object**

Var é utilizado para substituir o nome de um tipo, será do tipo do primeiro valor atribuído.

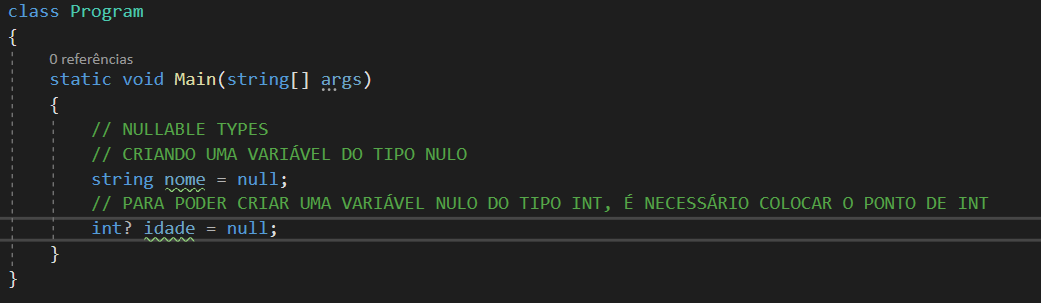


Object, como o próprio nome já disse ele é de um tipo objeto que recebe qualquer valor ou objeto, é utilizando quando se tem uma variável e você não sabe de qual tipo ela vai ser, neste caso pode ser utilizado o Object. A desvantagem dele é que não tem intelisense (ajuda dos editores).



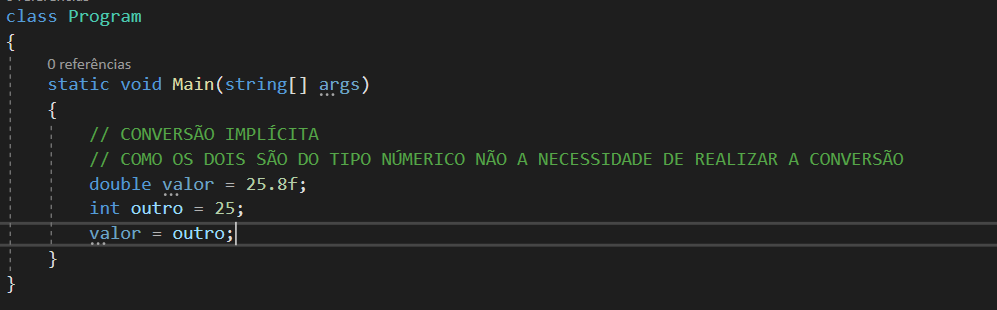
**Nullable Types**

Significa vazio, nulo, é diferente de zero ou uma string vazia. Todo tipo de dado primitivo ou complexo pode receber o valor null.



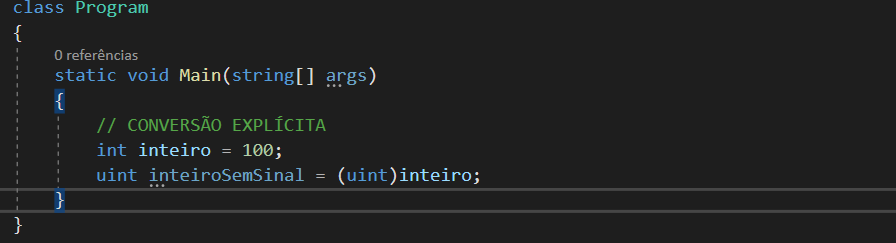
**Conversão de dados implícita**

É utilizada para transformar um dado de tipo X para tipo Y, existem dois tipos de conversão, as implícitas e as explicitas. As implícitas são conversões que podem ser executadas apenas com a passagem de dados e possuem tipos compatíveis.



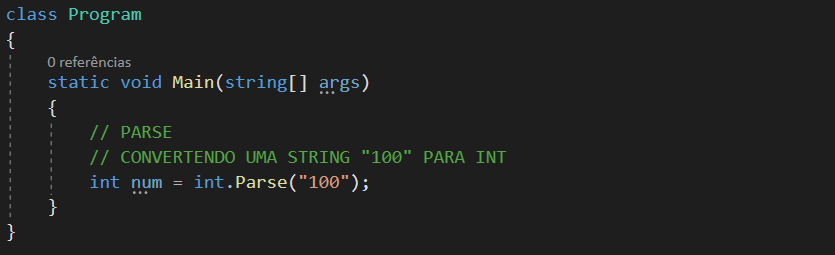
**Conversão de dados explicita**

O corre quando os tipos não são compatíveis, é dada pelo uso do tipo entre parênteses antes da atribuição.



**Parse**

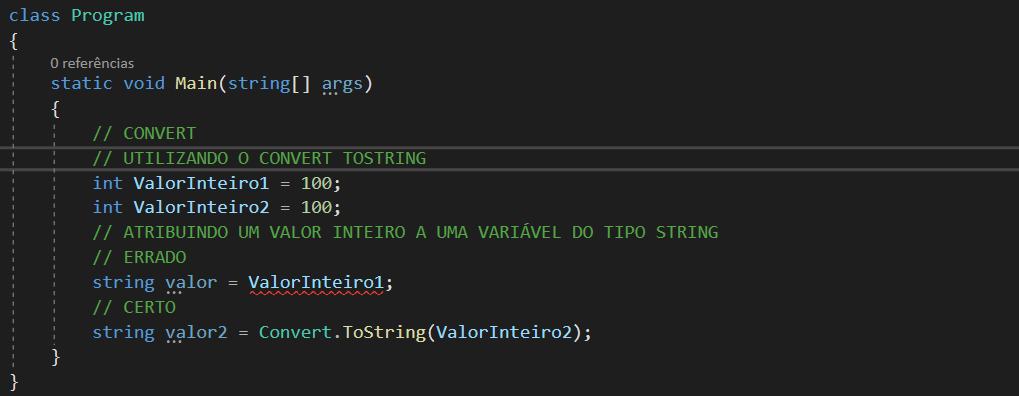
É usado extensão usada para converter um caractere ou string para um tipo qualquer.



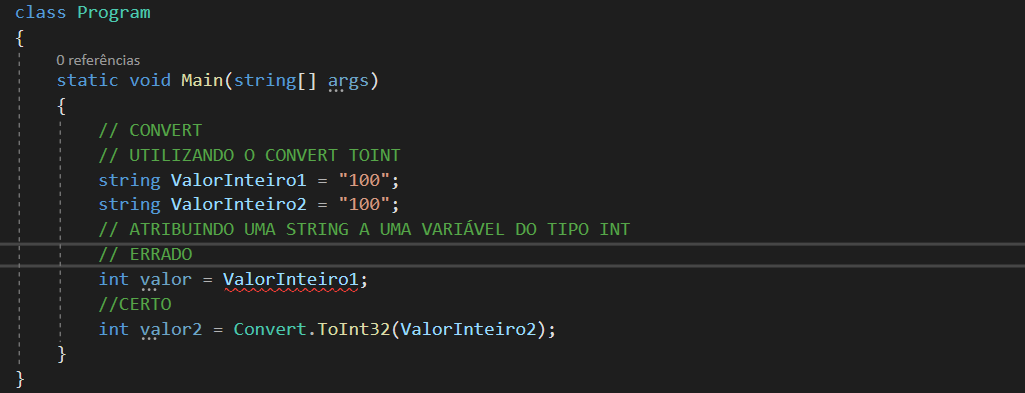
**Convert**

Similar ao parse visto anteriormente, porém permite converter vários tipos de valor. Devemos informar o tipo na chamada da conversão.

**Convertendo inteiro para string**

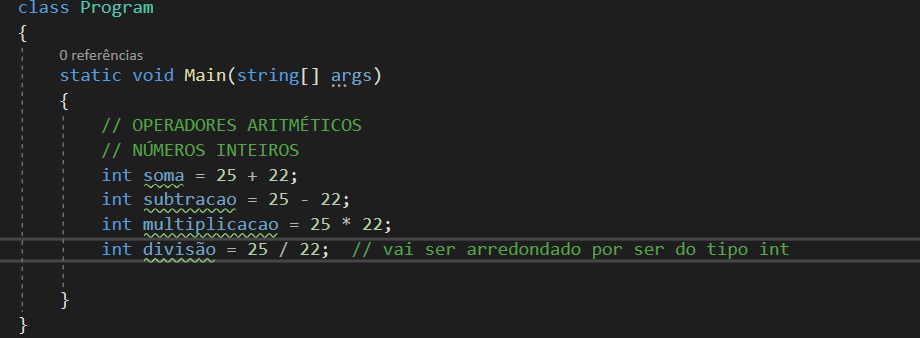


**Convertendo string para inteiro**



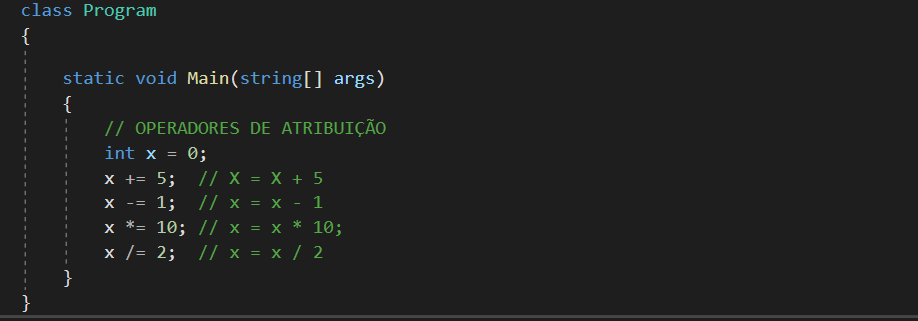
**Operadores aritméticos**

São operadores matemáticos dentro do c#, soma, subtração, divisão e multiplicação.



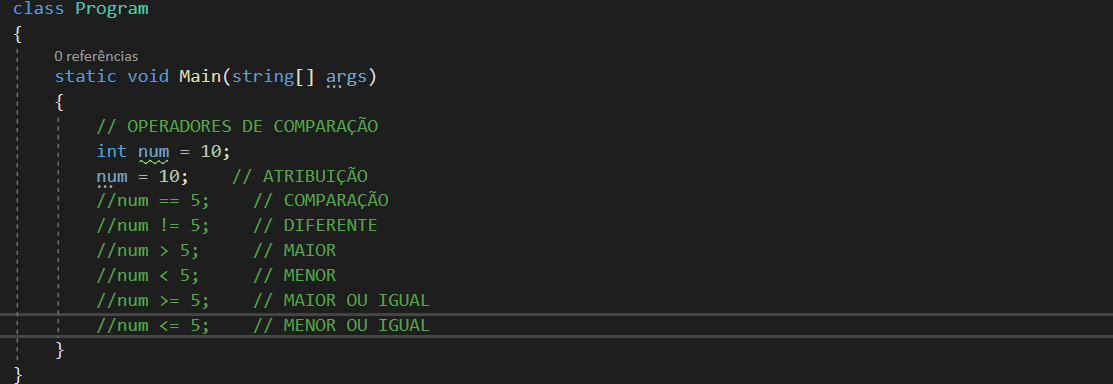
**Operadores de atribuição**

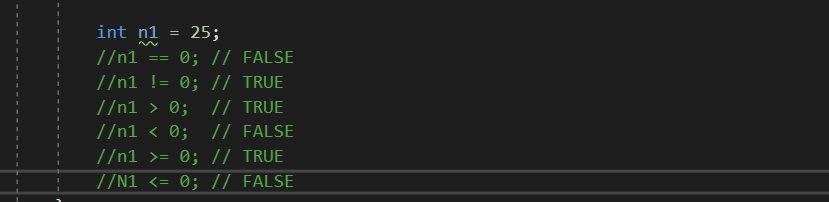
É utilizado quando queremos atribuir um valor a uma variável.



**Operadores de comparação**

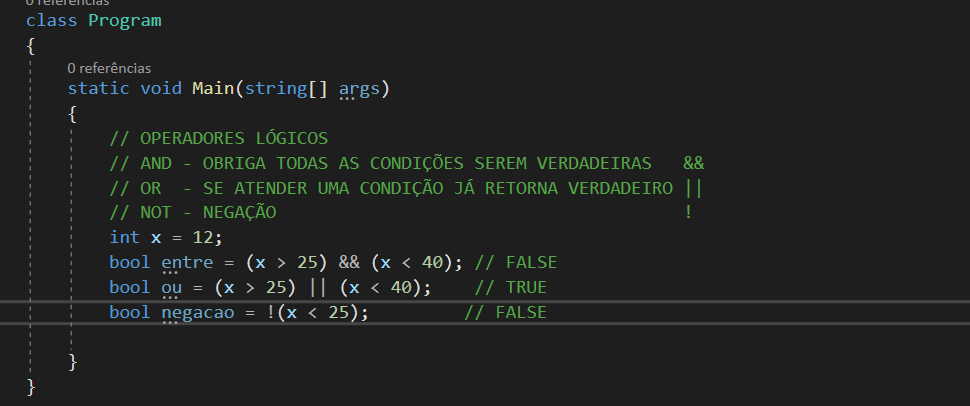
É utilizado para comparar um dado com o outro, podendo ser números, strings, bytes. A comparação sempre retorna verdadeiro ou falso (True ou False).





**Operadores lógicos**

São utilizados para executar operações condicionais para executar ou não algo, sempre retornando verdadeiro ou falso (True ou False).

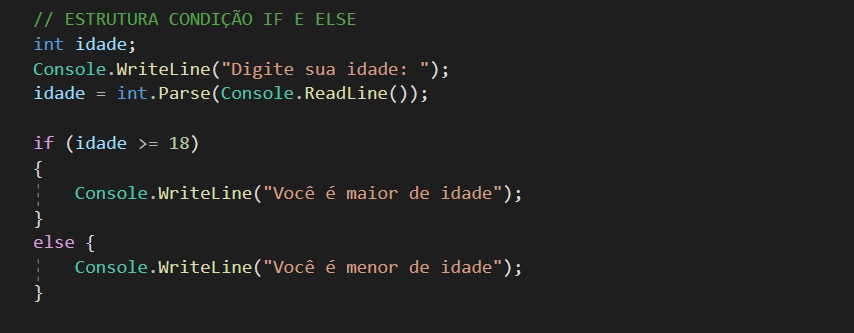


**Estrutura de condição IF e ELSE**

É utilizado para verificar uma ou mais condições, retornando verdadeiro ou falso.

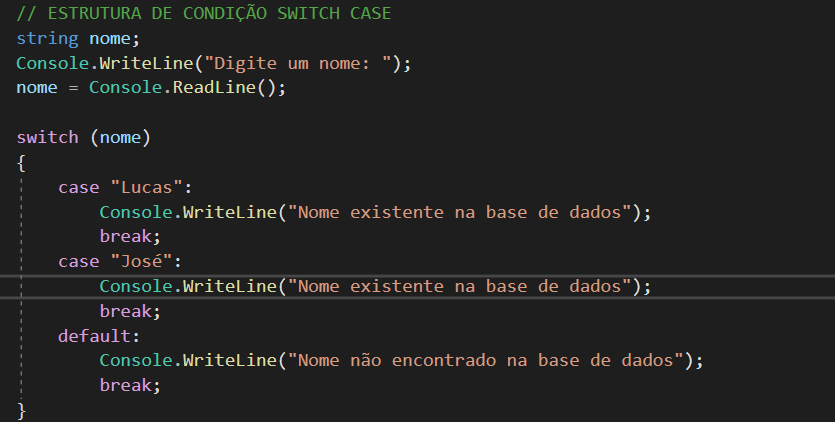
Exemplo:

Se a variável x for igual a 10, retornar true, caso contrario false.



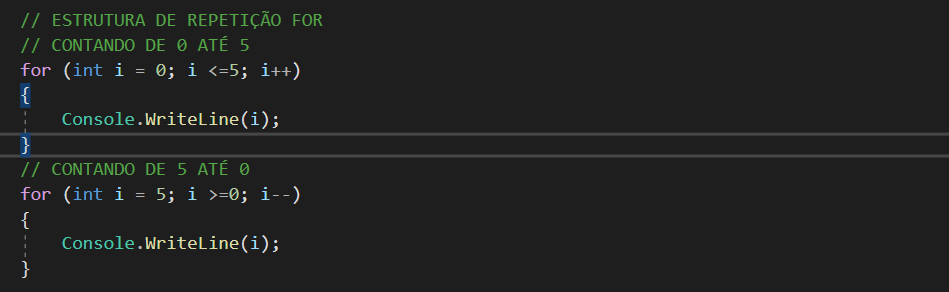
**Estrutura de condição Switch Case**

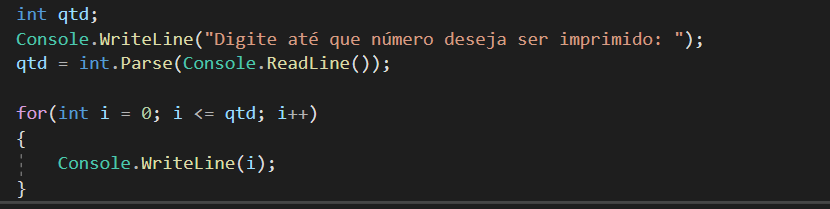
É uma forma de reduzir a complexidade de vários if e else encadeados. É muito utilizado, principalmente para uso em estruturas de menu. O conteúdo de uma variável é comparado com um valor constante, e caso a comparação seja verdadeira, um determinado comando é executado.



**Estrutura de repetição for**

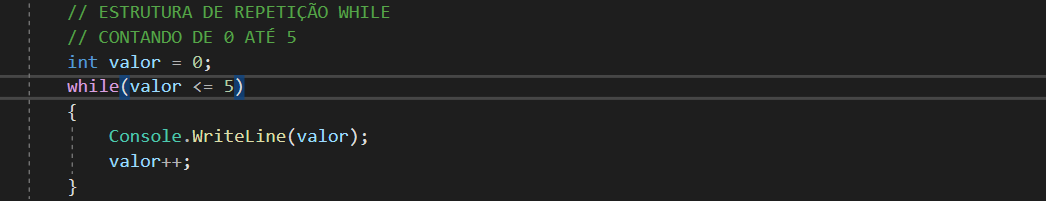
Na tradução “Para”, é um laço de repetição muito utilizado, executa determinada ação várias vezes que é definida por três parâmetros (contador, condição e incremento).





**Estrutura de repetição while**

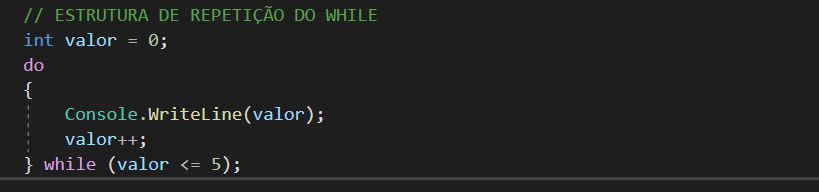
Na tradução “Enquanto”, ou seja, enquanto uma condição for verdadeira ele executa o código.



No começo do while a condição para executá-lo é enquanto a variável “valor” é menor ou igual a 5, ou seja, ela é true. Mas cada execução ela recebe o valor dela + 1, então após executar 6 vezes consecutivas o a variável “valor” vai ser maior que 5, que consequentemente vira uma condição do tipo false paralisando o while.

**Estrutura de repetição do-while**

Na tradução “Faça”, ou seja, realiza uma ação enquanto, checando a condição depois de executar diferente do while que verifica no início.



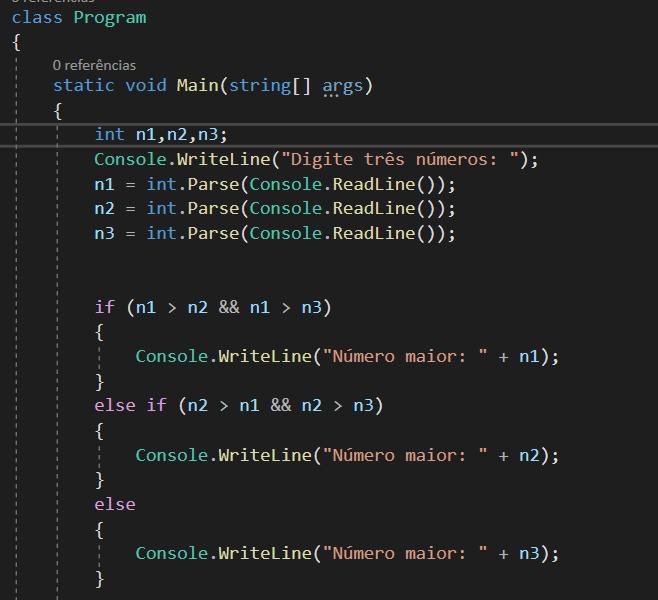
**Métodos e funções**

É um conjunto de comandos que realiza uma “função” específica, ou seja, ela é um pequeno “programa” utilizado por outros programas.

**Exemplo na prática:**

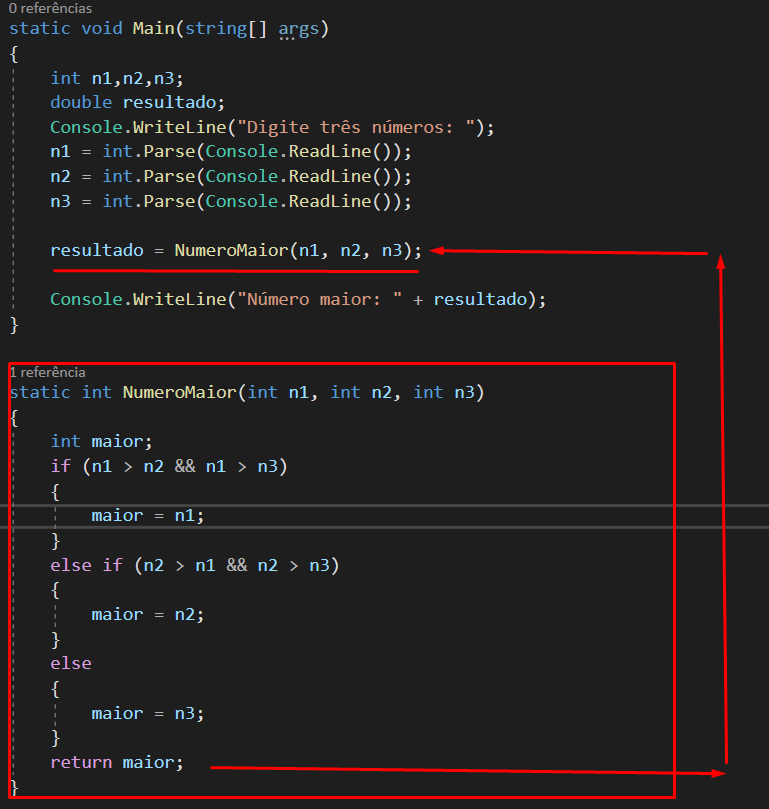
1 – Criar um programa que leia três números e mostre o maior número digitado

Na imagem abaixo, o exercício foi realizado sem a utilização da função, a desvantagem é que toda vez que for ler três números e mostrar o maior número digitado, vai ter de fazer novamente a verificação utilizando if, else.



Na imagem abaixo, foi utilizado a função criada “NumeroMaior”, para fazer a verificação, caso em outra parte do código tiver novamente está verificação, basta chamar a função e passar os parâmetros de entrada.

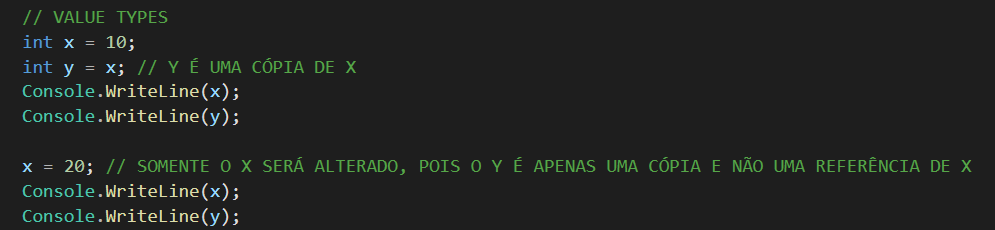
NumeroMaior (n1, n2, n3)



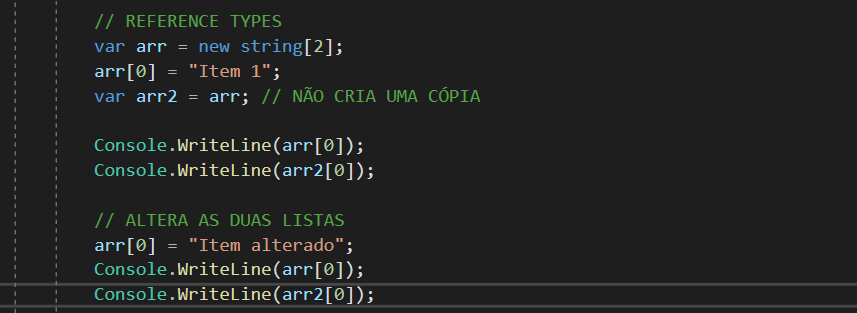
A variável resultado está recebendo o valor retornado da função “NumeroMaior”, que está passando como parâmetro de entrada os valores n1, n2, n3. Esses parâmetros estão sendo verificado na função qual deles é o maior e logo em seguida sendo retornado “return maior” que é recebido pelo resultado.

**Value Types e Reference Types**

Quando armazenamos um valor, a memória é alocada, este espaço armazena o dado criado e nossa variável acessa este dado diretamente. Caso criarmos uma variável do tipo de valor a outra o valor será copiado e ambas as variáveis serão independentes.



Y contínua com o valor 10, pois é uma cópia armazenada no stack da memória onde não temos referência. Diferente do Reference Types eles armazena o endereço do objeto que contém os dados, não armazena os dados em si, a referência é armazenada em um local da memória chamado de Heap.



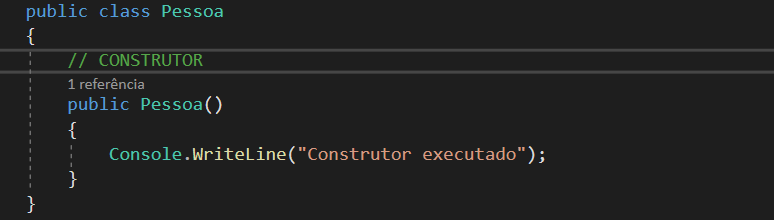
Classes, objetos e arrays são dos tipos Reference Types.

**Construtor**

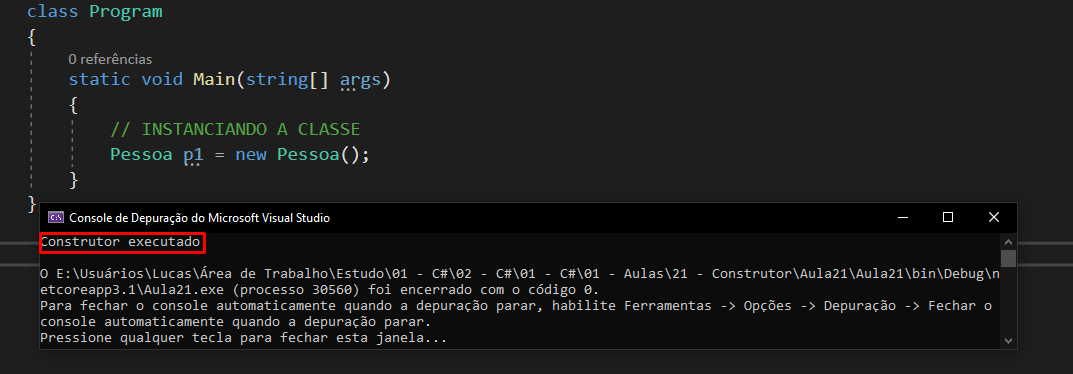
É um método com o mesmo nome da classe, ele é executado assim que instanciamos um objeto da classe.

Exemplo:

Foi criada uma classe “Pessoa” com um método construtor para que no momento que a classe é instanciada mostre uma mensagem.



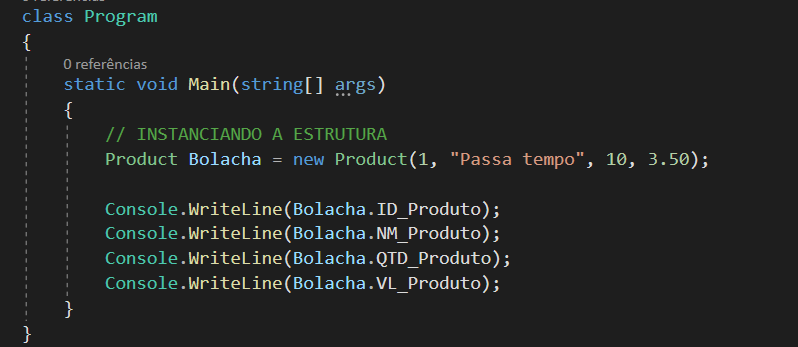
Na imagem abaixo instanciaremos a classe “Pessoa” para que seja exibido a mensagem do método construtor.



**Structs**

Structs são um tipo de valor, ele é um value type e são considerados como semântica de valor. Exemplificando ele serve para estruturar os nossos dados, ele também é composto de propriedades e métodos.

Na primeira imagem está sendo instanciado o método construtor, passando os seus parâmetros. Em seguida está sendo impresso os valores que estão em cada atribuído.



Na segunda imagem foi criado um struct com seus atributos e métodos, também foi criado um método construtor que recebe os parâmetros passados na primeira imagem e atribui aos atributos criados do struct.

