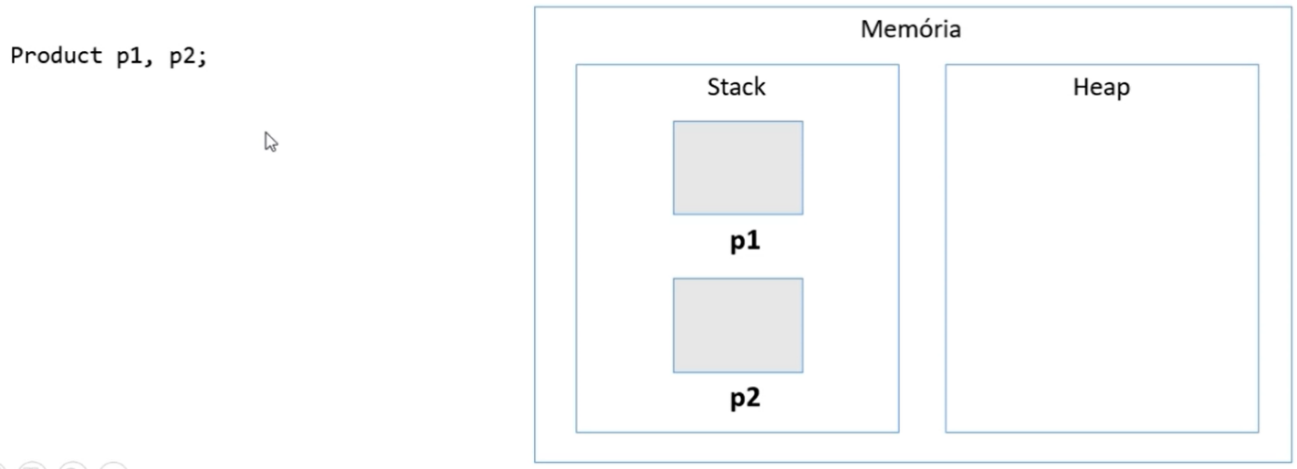
# Tipos referência vs. tipos valor 18/11/2020

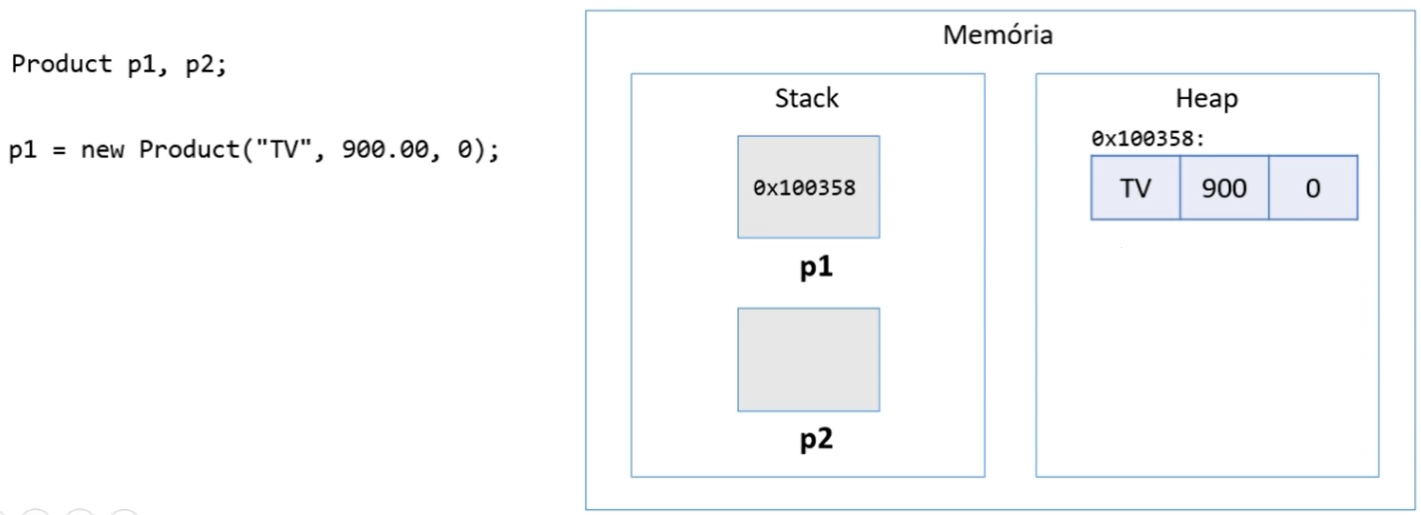
# Classes são tipos referência

Variáveis cujo tipo são classes não devem ser entendidas como caixas, mas sim como “tentáculos” (ponteiros) para caixas.

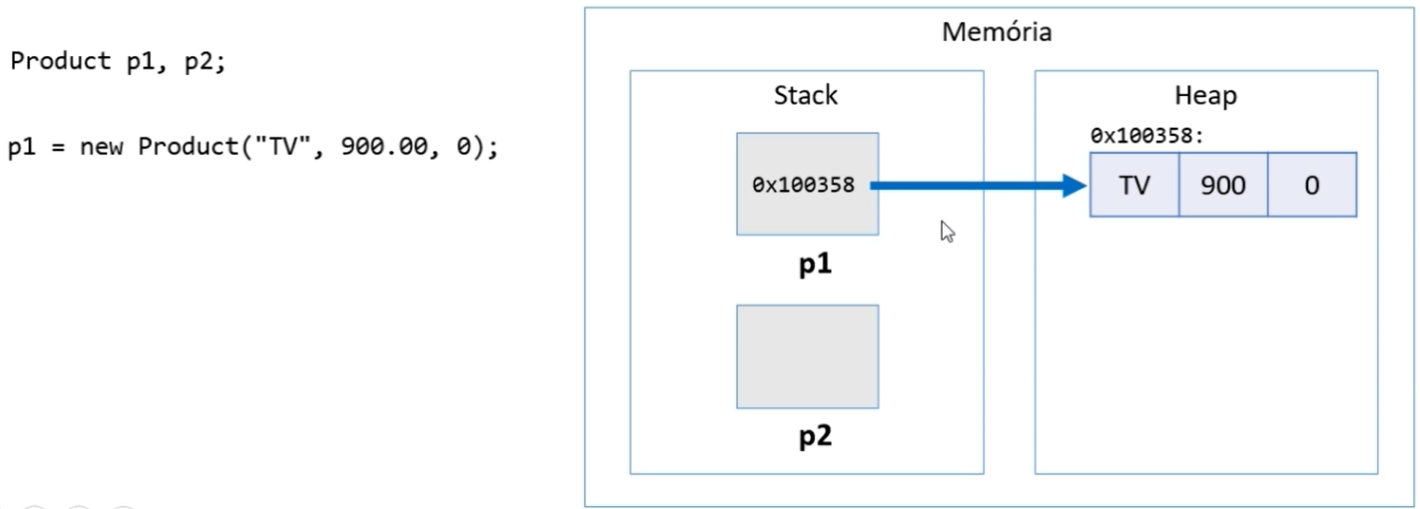
Ao declarar as variáveis. Elas são alocadas em um lugar da memória chamada Stack.



Quando a variável é instanciada. São criadas “caixas” onde os valores são armazenados. Na variável p1, fica apenas a posição da memória onde o valor foi armazenado.

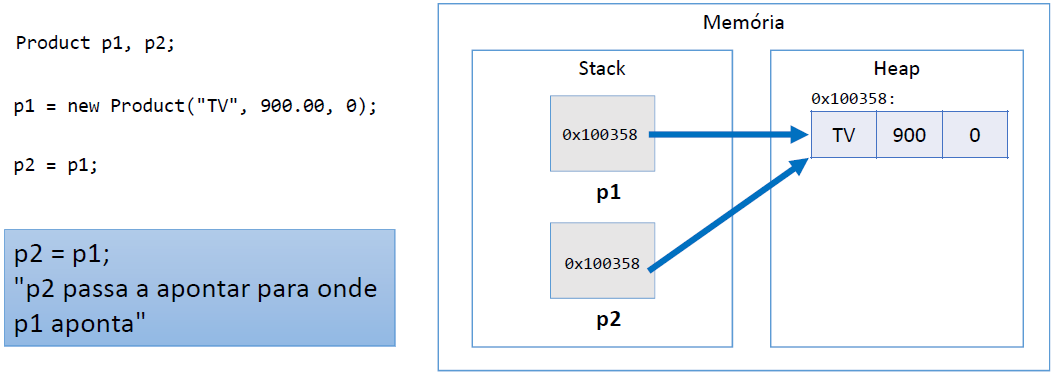


Utilizamos o termo ponteiro, porque a variável, armazena apenas o endereço na memória de onde o valor foi armazenado.

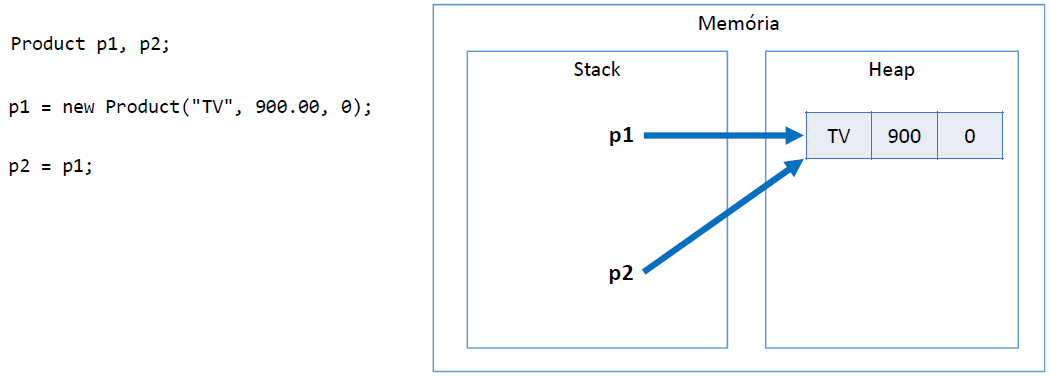


Quando uma variável recebe outra. As duas variáveis estarão apontando para o mesmo endereço da memória.

Por disso, podemos afirmar que, **classes são tipos referências.**

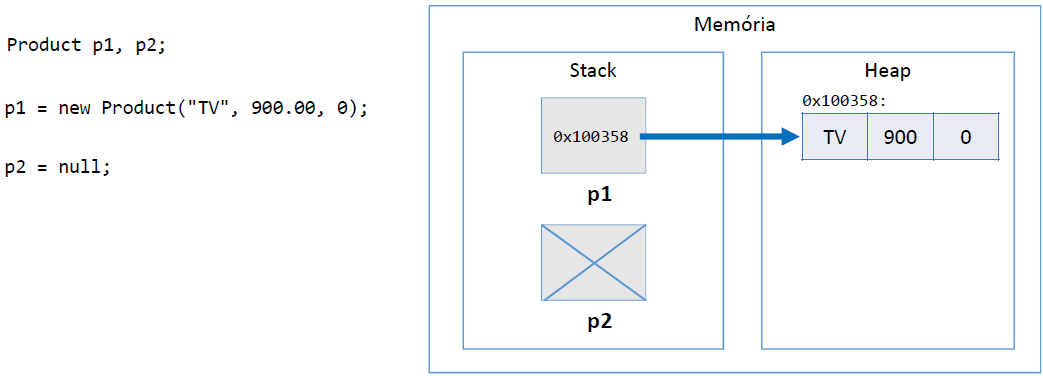


Desenho simplificado. O mais utilizado em bibliografias.



# Valor "null"

Tipos referência aceitam o valor "null", que indica que a variável aponta para ninguém.

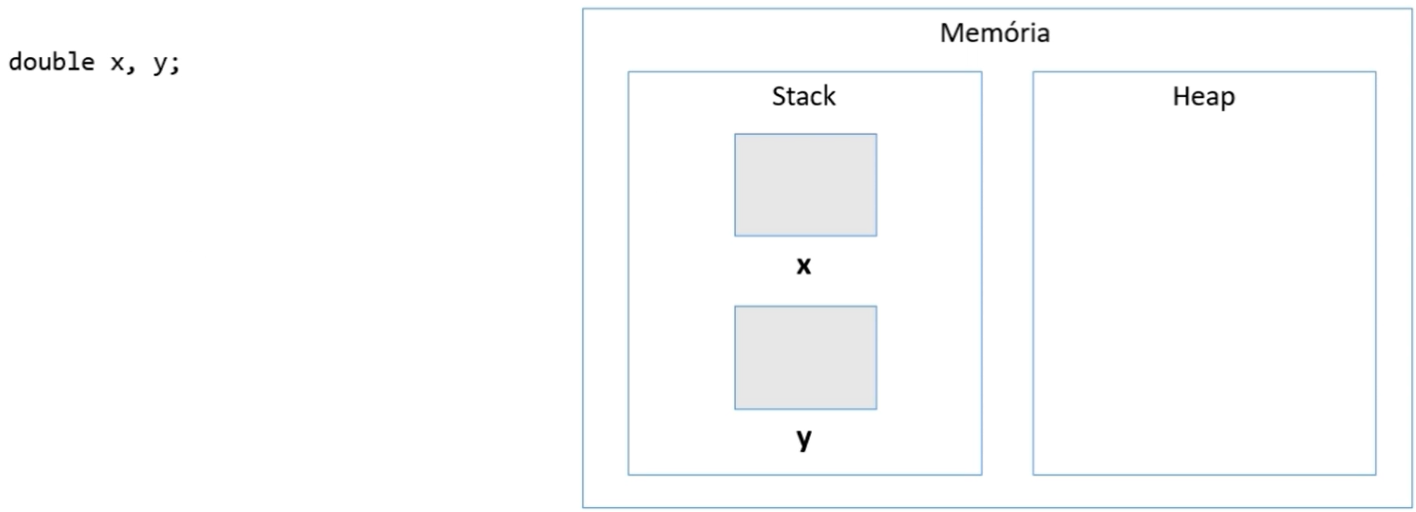


# Structs são tipos valor

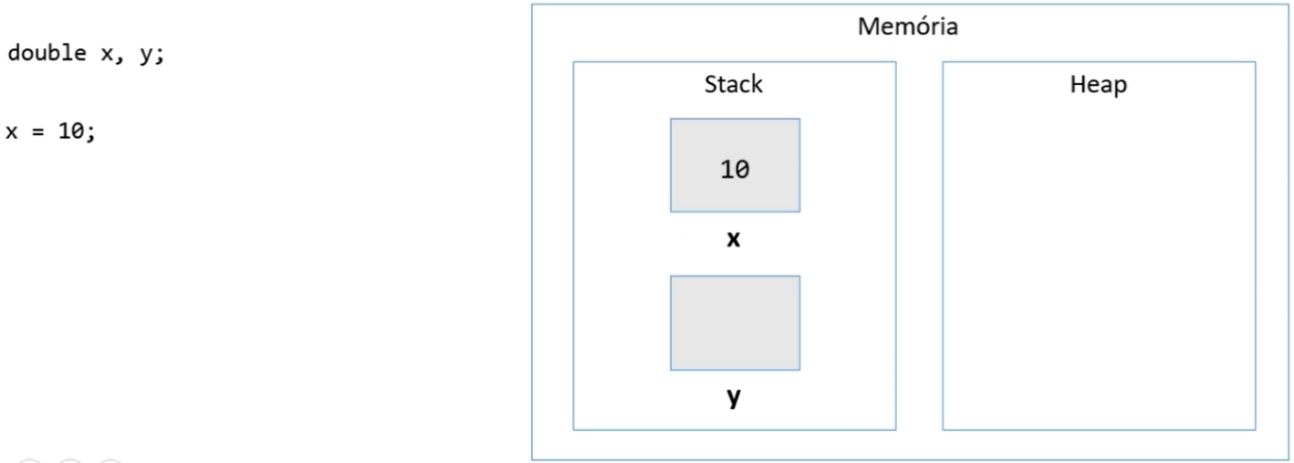
Structs, são os tipos básicos da linguagem. Como double, int, float, ....

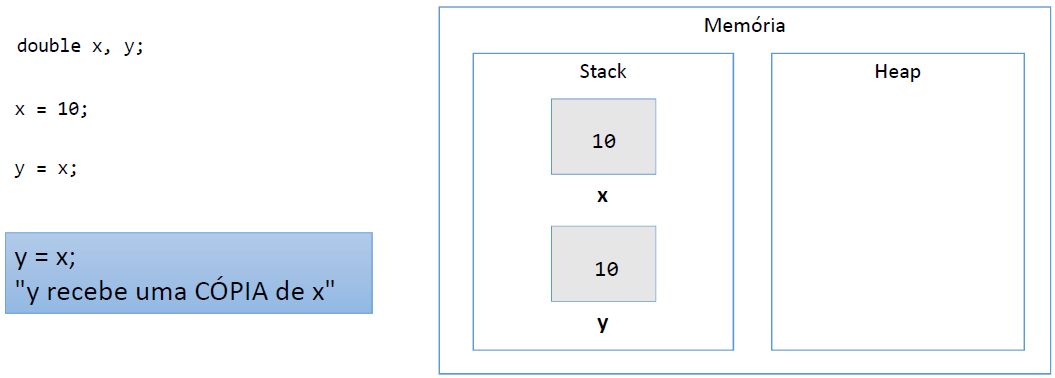
A linguagem C# possui também tipos valor, que são os "structs". Structs são “CAIXAS” e não ponteiros.

Quando declaramos duas variáveis do tipo struct. Duas “caixas” são criadas.



Quando uma variável recebe um valor. Exemplo:

Quando uma variável recebe outra variável do tipo struct. Não é criado uma seta. A variável recebe uma cópia do valor. Dai, temos **Struct tipo valor.**



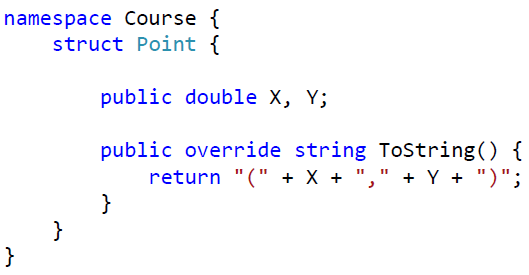
Tipos structs – Tipos valor



Outros structs importantes: DateTime, TimeSpan (um estante e uma duração)

# É possível criar seus próprios structs

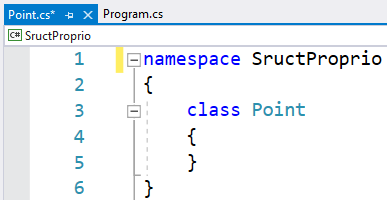
Nesse exemplo, vamos criar um ponto em um plano cartesiano. Bem parecido com a classe, mas com alguns recursos a menos. Suas variáveis são armazenadas no Stack da memória.



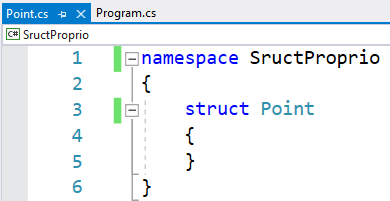
# Structs e inicialização

Demo:

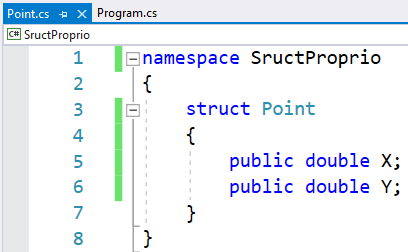
Vamos criar uma classe com o nome Point:



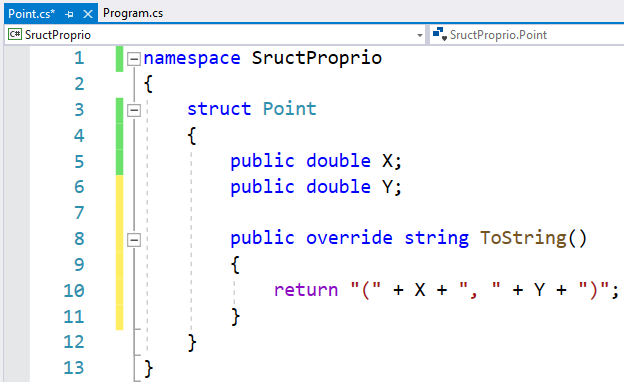
Altere o class por struct



Vamos criar os atributos da classe



Vamos criar um override



Código da classe Point:

namespace SructProprio

{

struct Point

{

public double X;

public double Y;

public override string ToString()

{

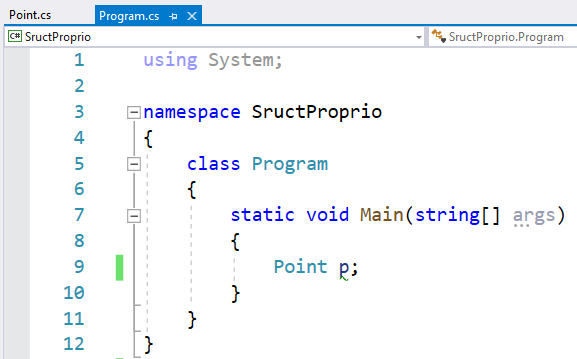
return "(" + X + ", " + Y + ")";

}

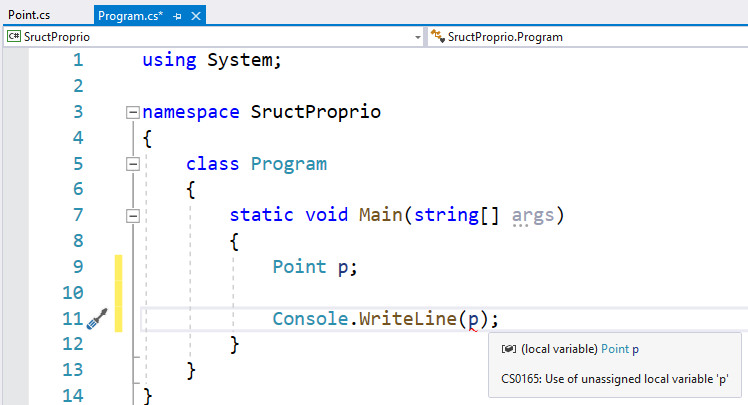
}

}

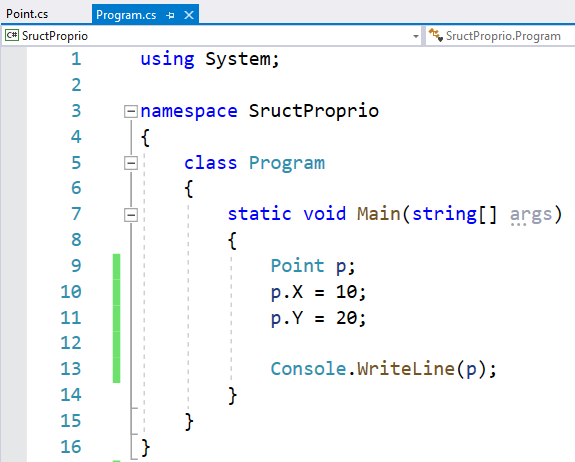
Agora no programa principal. Vamos declarar uma variável do tipo Point.



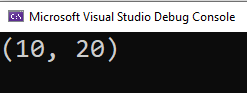
Mesmo o p sendo um tipo struct, o compilador não permite que eu imprimar esse valor. Ele diz que a variável não foi inicializada.



Portando, para utilizar o struct, precisamos inicializa-la. Atribuir um valor.



Resultado.

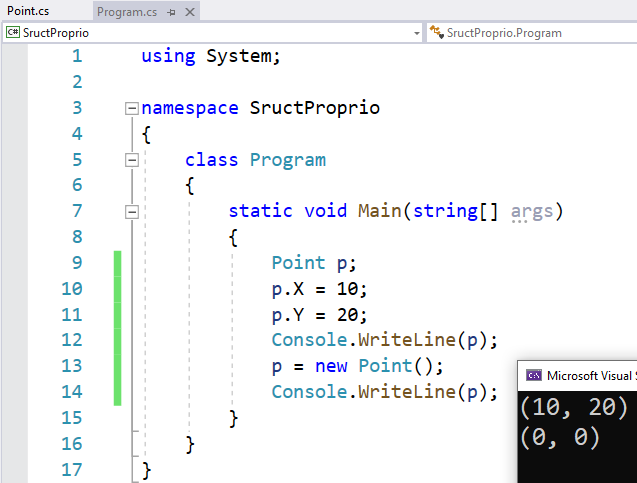


Pode inicializar atribuindo valores diretamente ou com a inicialização new.

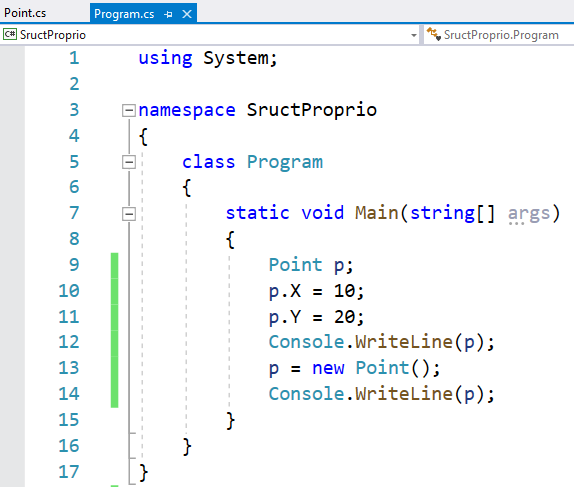
Point p = new Point(); // Opcional

Porque é um tipo struct. Ao iniciar, ele já cria a “caixa” no stack da memória.

Entretanto, o struct aceita a instanciação com o new. Exemplo.



Foi recriado o p, mas com valores 0 (zero). Ou seja. Mesmo sendo struct. É necessário inicializar.



Código da classe principal

using System;

namespace SructProprio

{

class Program

{

static void Main(string[] args)

{

Point p;

p.X = 10;

p.Y = 20;

Console.WriteLine(p);

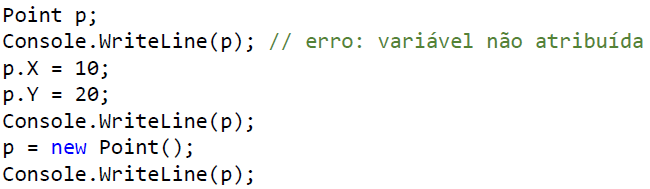
p = new Point();

Console.WriteLine(p);

}

}

}

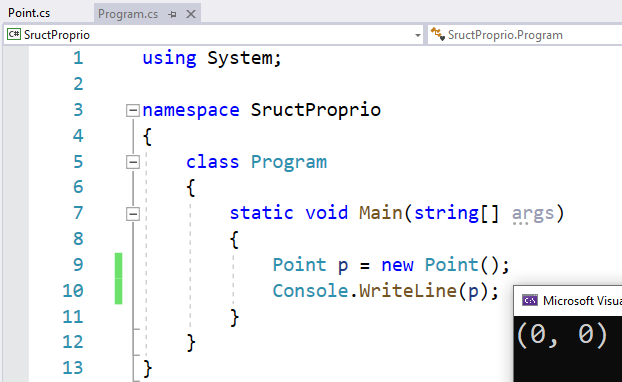


# Valores padrão

Quando alocamos (new) qualquer tipo estruturado (classe, struct, array), são atribuídos valores padrão aos seus elementos

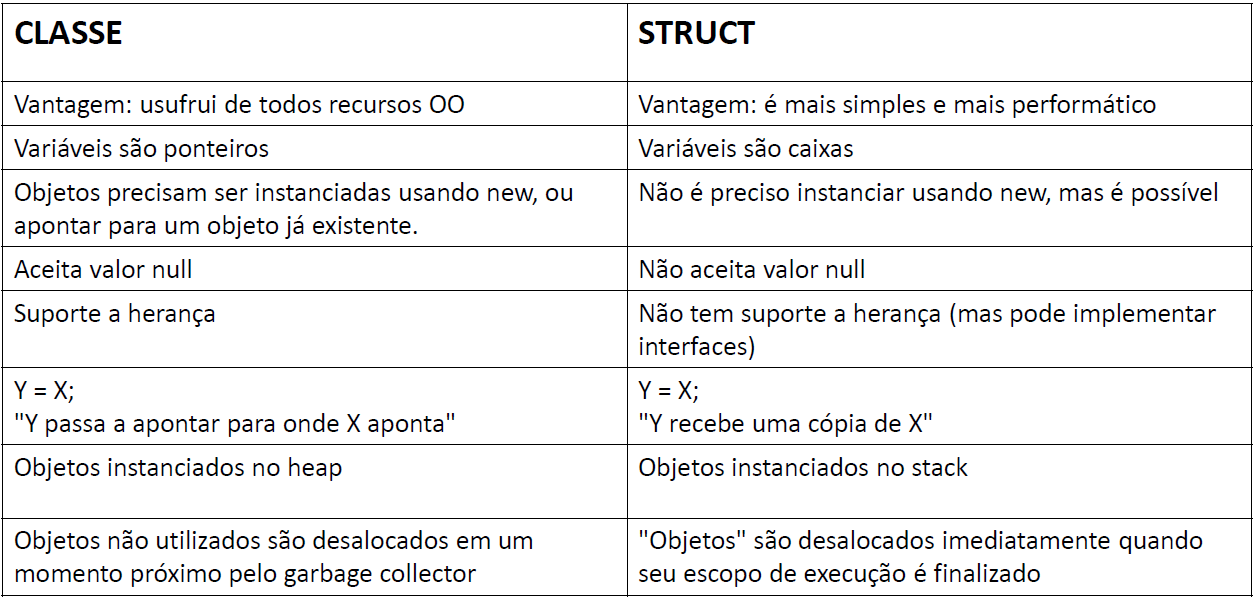
* números: 0
* bool: False
* char: caractere código 0. (Não é o caracter 0 zero. É o código 0).
* objeto: null

Exemplo:



Lembrando: uma variável apenas declarada, mas não instanciada, inicia em estado "não atribuída", e o próprio compilador não permite que ela seja acessada.

# Tipos referência vs. tipos valor



Importante:

O struct é mais simples, mais performático.

Exemplo: Caso precise trabalhar com uma matriz muito grande para receber pixel de imagens. Trabalhar com valores struct irá proporcionar um ganho de velocidade.