**CLASSE**:

Definição de um tipo

No exemplo do exercício a classe Triangulo define que o triangulo é um cara que possui os dados A, B e C do tipo double.

**OBJETOS**

São instancias da classe (casos particulares daquela classe)

No caso da classe Triangulo, ela possui dois objetos (x e y)

**ATRIBUTOS**

São as campos do objeto.

No caso da classe Triangulo, os atributos de x e y são A, B e C.

Nomes de Atributos iniciam com letra maiúscula.

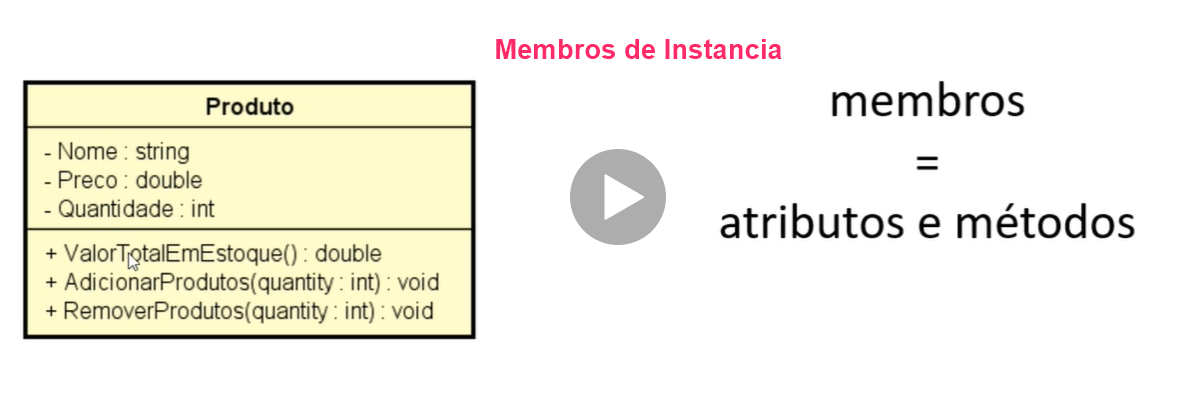
**MÉTODO**

São as operações/funções

Também é um membro da classe. Nome inicia com letra maiúscula.

**MEMBROS**

Atributos e Métodos



MEMBROS ESTÁTICOS

Também são membros de classe

São membros que fazem sentido independente do objeto. Não precisam de objeto para serem chamados, são chamados a partir do próprio nome da classe.

\*\*Não é possível chamar uma função que não é estática dentro de uma função estática da mesma classe, isto é, se tem uma função estática dentro da classe e ela precisa chamar outra função, essa outra função precisa ser estática tbm\*\*

Aplicações Comuns:

* Classes Utilitárias: Math.Sqrt(Double);
* Declaração de Constantes;

**SOBRECARGA**

Recurso que uma classe possui de oferecer mais de uma operação com o mesmo nome, porém com diferentes listas de parâmetros.

Toda classe no C# é uma subclasse da classe Object

**ToString** – Converte o objetivo para String

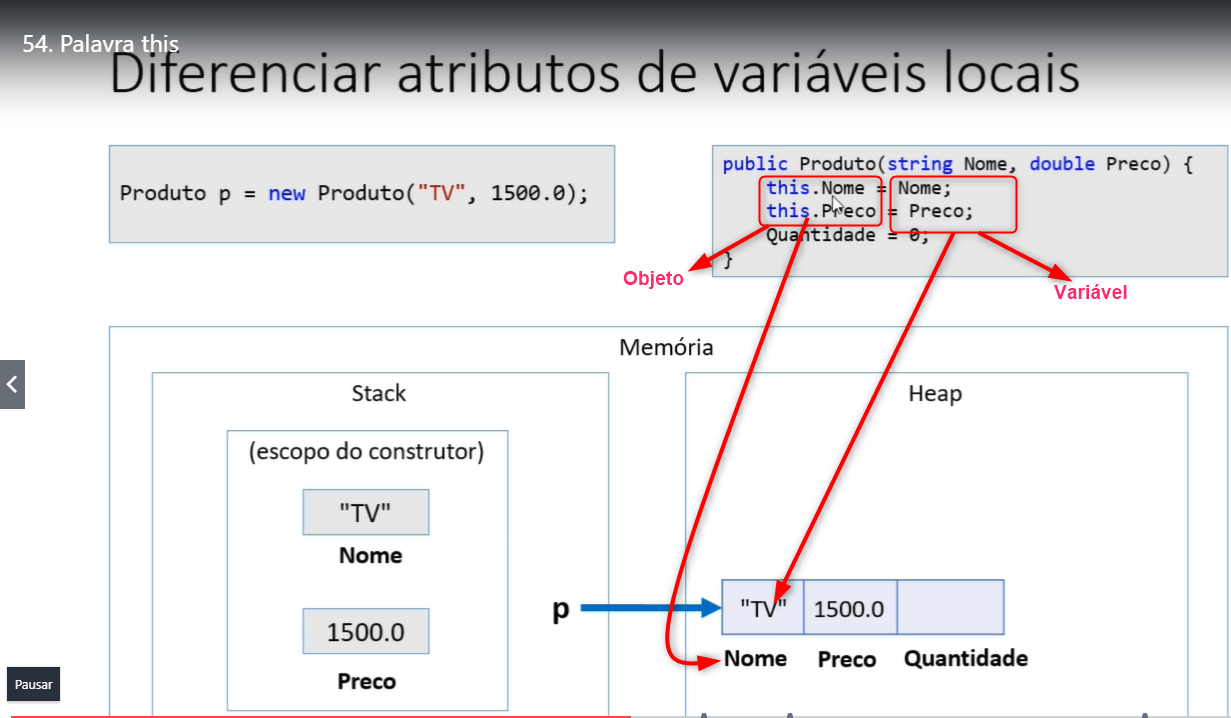
Variável de entrada de parâmetro de função começa com letra minúscula;

**Sintaxe Alternativa para Iniciar Valores**

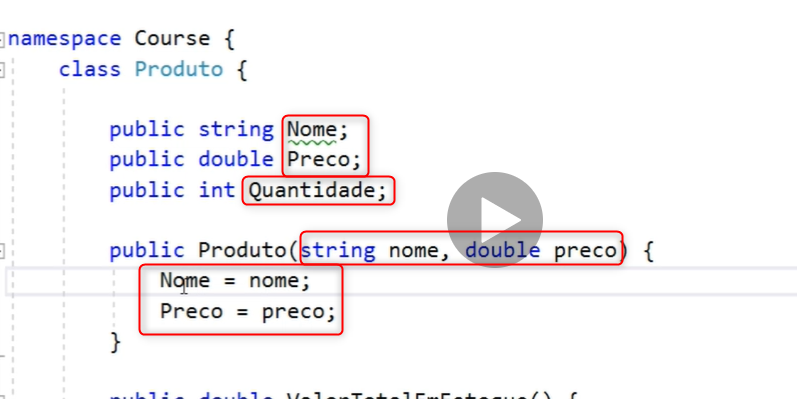
**Palavra this**

Uma das suas funções é diferenciar os atributos das variáveis locais, conforme imagem abaixo:

Na linguagem C# isso não é muito comum porque usamos o padrão de nomes onde a variável recebe a primeira letra minúscula e o atributo é sempre com letra maiúscula (conforme imagem 2)

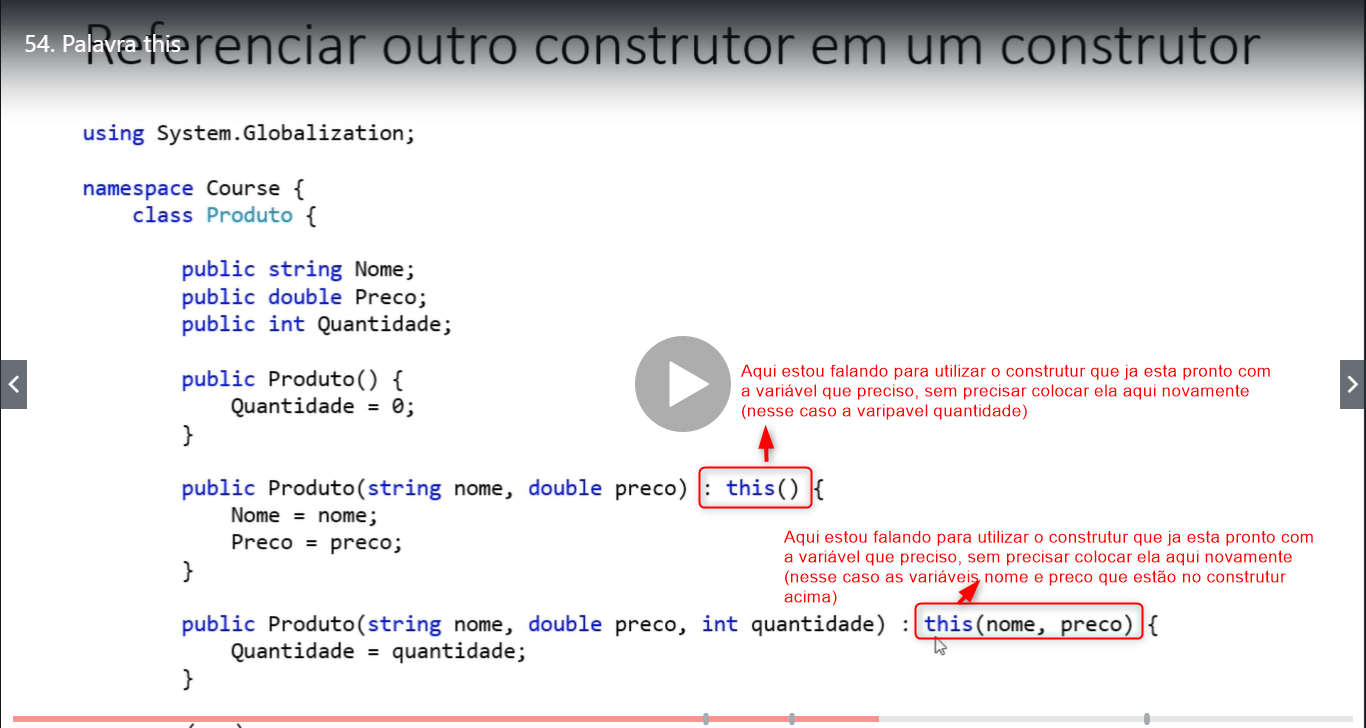


*Imagem 1*



*Imagem 2*

Outra utilidade da palavra This, é quando vc vai referenciar outro construtor em um construtor (exemplo no exercício newProduto4)



Outra utilidade da palavra this é passar o próprio objeto como argumento na chamada de um método ou construtor

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**ENCAPSULAMENTO**

O Encapsulamento protege detalhes da implementação, expondo apenas operações seguras.

//atributo private significa que o atributo não pode ser acessado por outra classe, isso é encapsulamento

//atributos private começam com \_ e letra minúscula

Para tornar possível acessar os atributos private, utilizar os comandos get e set.

Ma das vantagens do encapsulamento é que é possível incluir uma lógica no método (por exemplo, você pode tratar se não quer que uma variável não receba um conteúdo vazio, aí só colocar um IF, por exemplo)

Não é muito usual no C# porque o C# tem conceito de properties

**PROPRIEDADES**

São definições de métodos encapsulados, porém expondo uma sintaxe similar à de atributos e não de métodos.

**Auto properties**

É uma forma simplificada de se declara propriedades que não necessitam de lógicas particulares para operações get e set.

**Ordem sugerida para implementação de membros de uma classe**

1. Atributos privados;
2. Propriedades autoimplementadas;
3. Construtores;
4. Propriedades customizadas
5. Outros Métodos da Classe;

Tela de computador

Descrição gerada automaticamente

Tela de computador

Descrição gerada automaticamente

**Modificadores de acesso para membros de uma classe**

**Membros da Classe**

Tabela

Descrição gerada automaticamente

**A própria classe**

**Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente**

Quando o tipo da classe não está definida, ela é considerada como “Internal”

No Exemplo abaixo, somente outras classes dentro do mesmo projeto podem acessesá-lo.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**MODIFICADOR DE PARAMETRO REF E OUT**

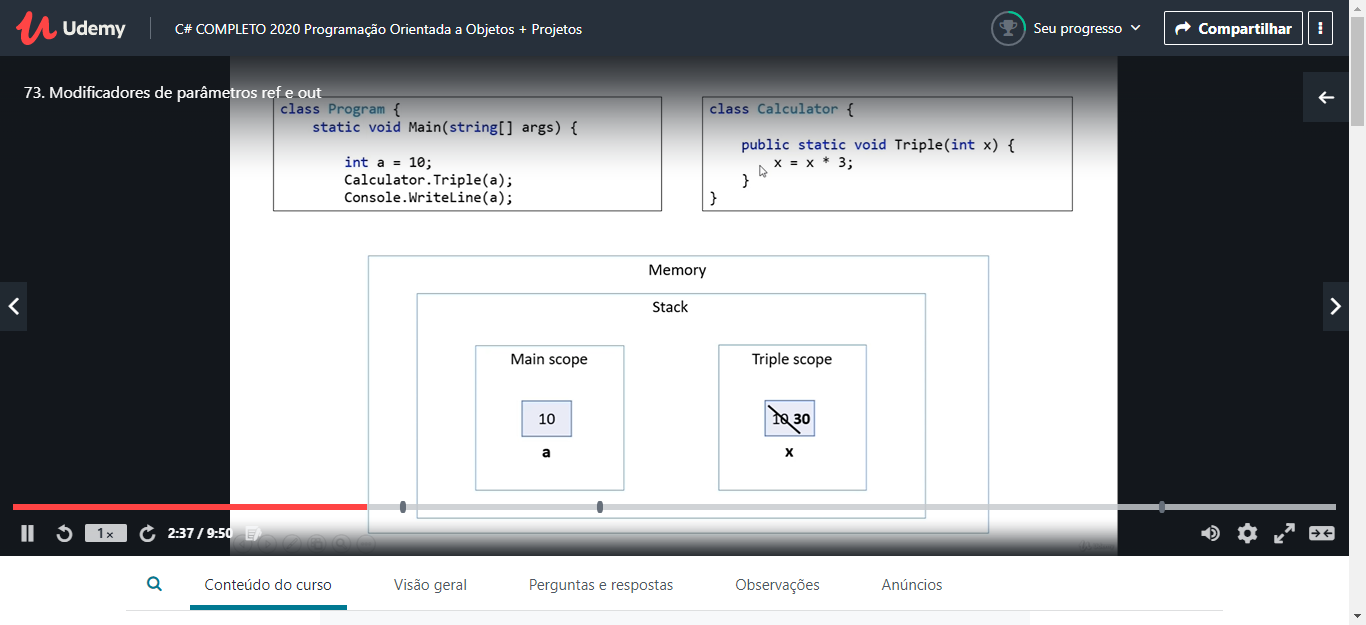
**São considerados “code Smells” e devem ser evitados**

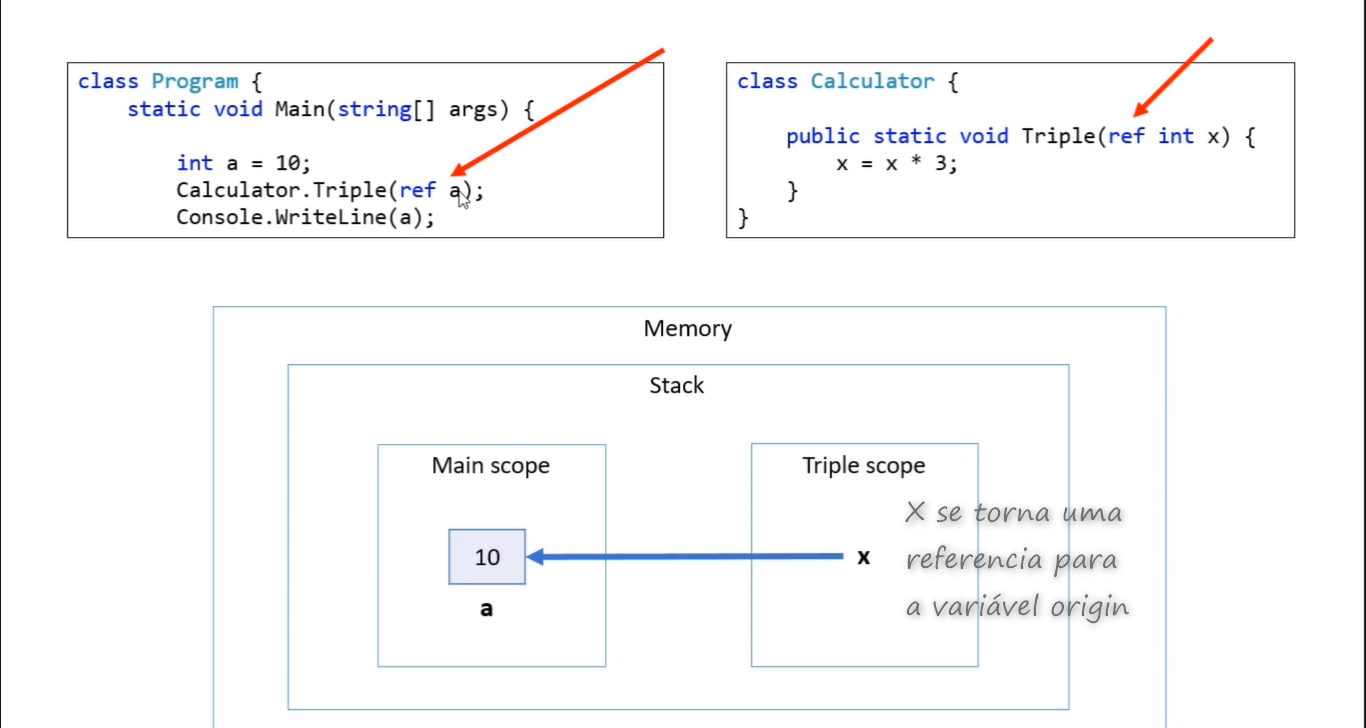
**\*acredita-se que esses modificadores existem pois no inicio o .Net tb aceitava c++ e ela lida com referencia de variável explicita.**

**REF**

Na situação abaixo em que o objetivo é triplicar um valor foi criada uma classe que realiza o cálculo, porém quando passa o parâmetro para a classe, o valor de X que será triplicado (variável da classe), e não a variável a que é do program. Para fazer essa referencia necessário usar o REF

\*O modificador ref exige que você inicie a variável;

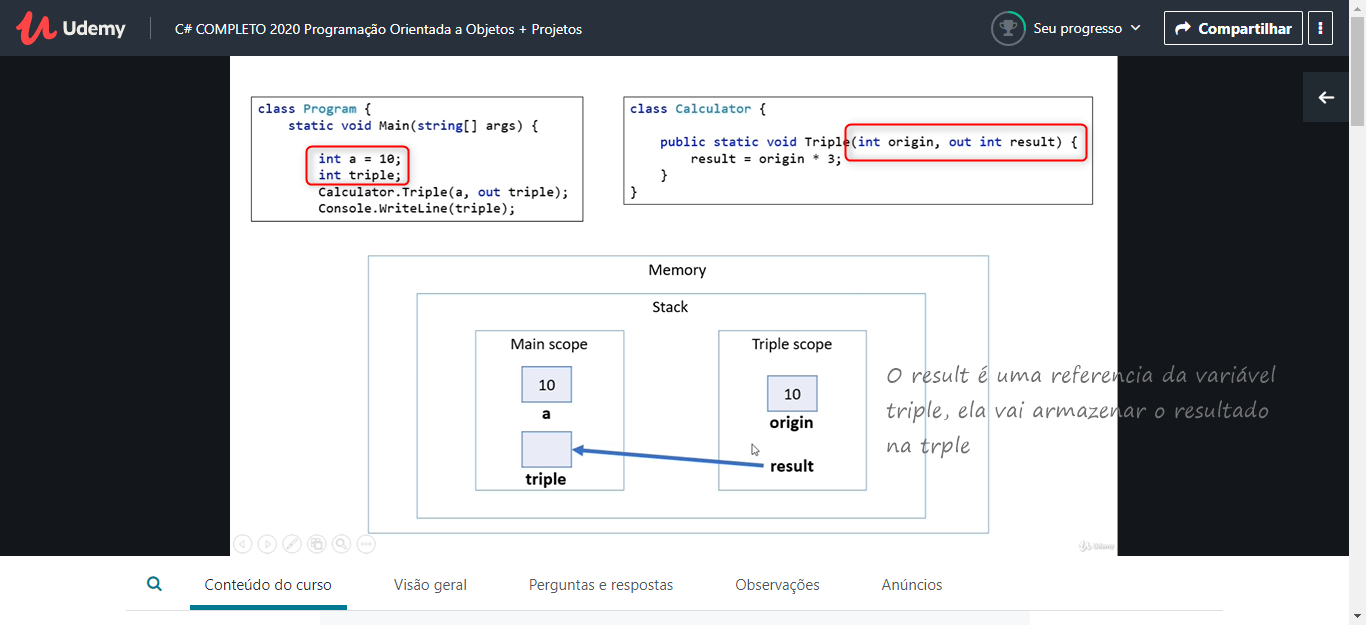




OUT

O modificador out não exige que você inicie a variável.

Trata-se de uma variável de saída.



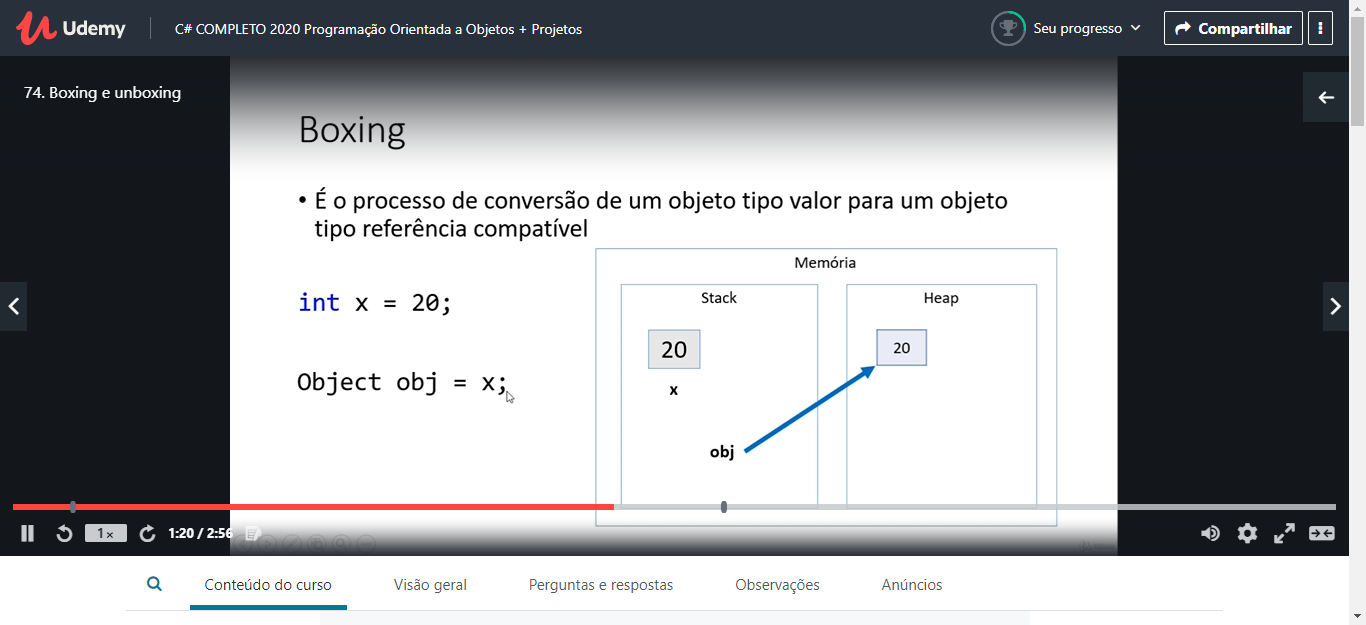
**BOXING E UNBOXING**

**Boxing** é um processo de conversão de um objeto tipo valor para um objeto tipo referencia compatível

Object é um tipo genérico do C#, ele é compativel com qqr tipo, ele é uma classe.

No caso abaixo o object “obj” vai ser referencia para um novo objeto que será instanciado no Heap:

É importante saber que utilizando o object tem o custo de memória pois ele é um objeto, não uma simples variável.



**Unboxing** é o processo de conversão de um tipo referencia para um objeto de valor compatível.

Nesse caso a variável recebe o conteúdo do objeto.

**SINTAXE OPCIONAL**

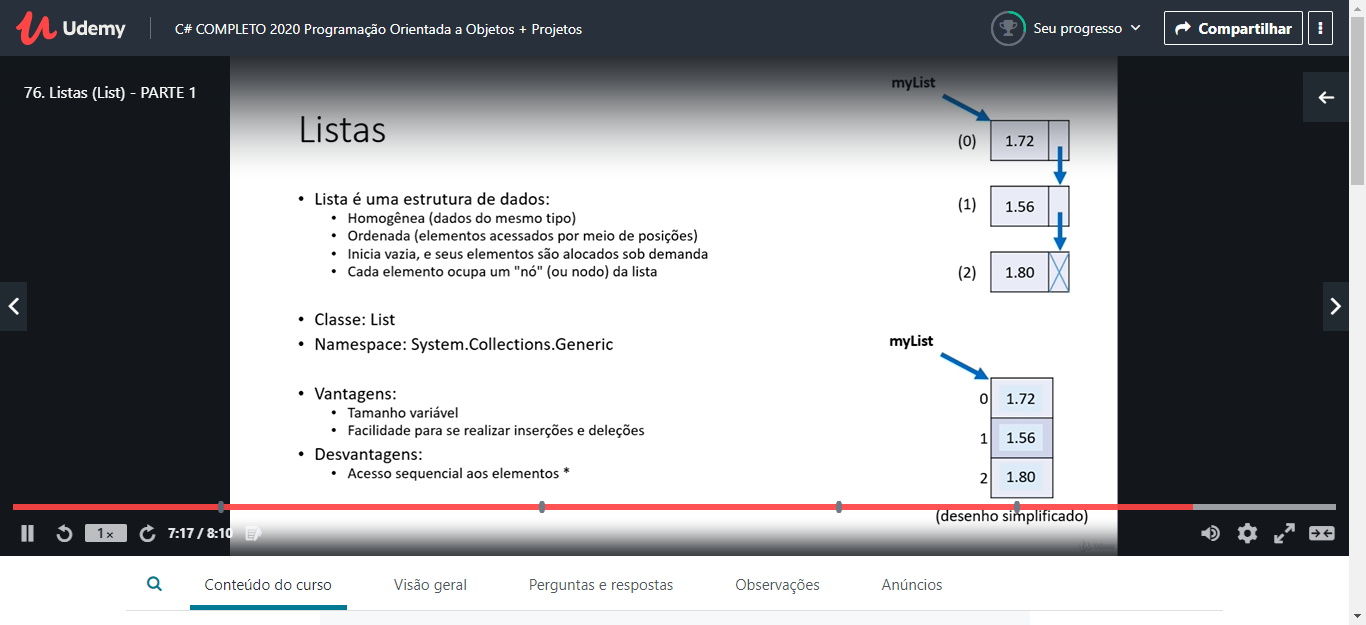
**Laço For Each** Sintaxe opcional e simples para percorrer coleções (vetores, listas, outros tipos de estrutura de dados). Exemplos na pasta de exercícios

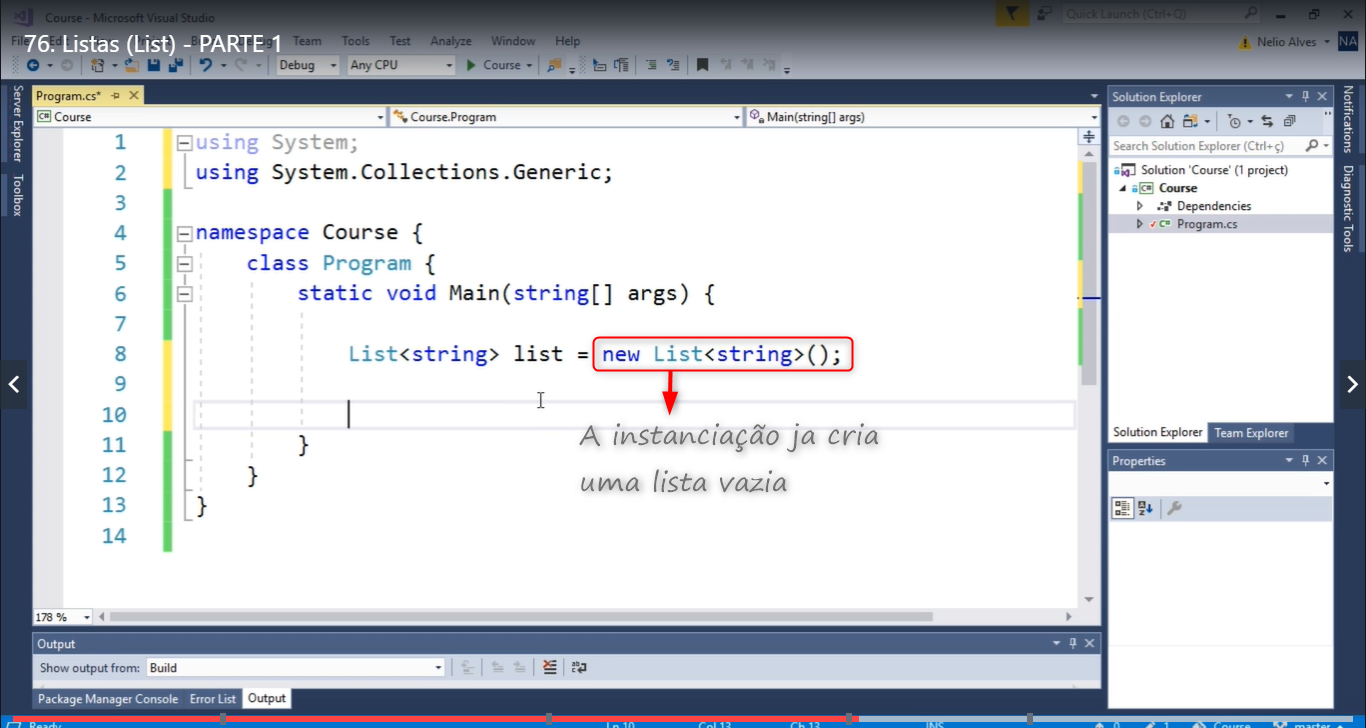
**LISTAS**

<https://msdn.microsoft.com/en-us/library/6sh2ey19(v=vs.110).aspx>

O conceito é parecido com vetores, mas os vetores precisam ser criados já com o numero especifico de elementos, já a lista é alocada sob demanda, isso significa que não precisa informar o numero de elementos que ela terá.

Nome da lista sempre em letra minúscula.

****

****

**No C# não é necessário definir o tipo da variável (Tipe Inference), mas por padrão importante definir.**