Funções (C# - CSharp)

Funções locais são métodos privados de um tipo que estão aninhados em outro membro. Eles só podem ser chamados do membro que os contém. Funções locais podem ser declaradas em e chamadas de:

- Métodos, especialmente os métodos iteradores e os métodos assíncronos
- Construtores
- Acessadores de propriedades
- Acessadores de eventos
- Métodos anônimos
- Expressões lambda
- Finalizadores
- Outras funções locais

No entanto, as funções locais não podem ser declaradas dentro de um membro apto para expressão.

Funções

- Representam um processamento que possui um significado
  - Math.Sqrt(double)
  - Console.WriteLine(string)
- Principais vantagens: modularização, delegação e reaproveitamento
- Dados de entrada e saída
  - Funções podem receber dados de entrada (parâmetros ou argumentos)
  - Funções podem ou não retornar uma saída
- Em orientação a objetos, funções em classes recebem o nome de "métodos"

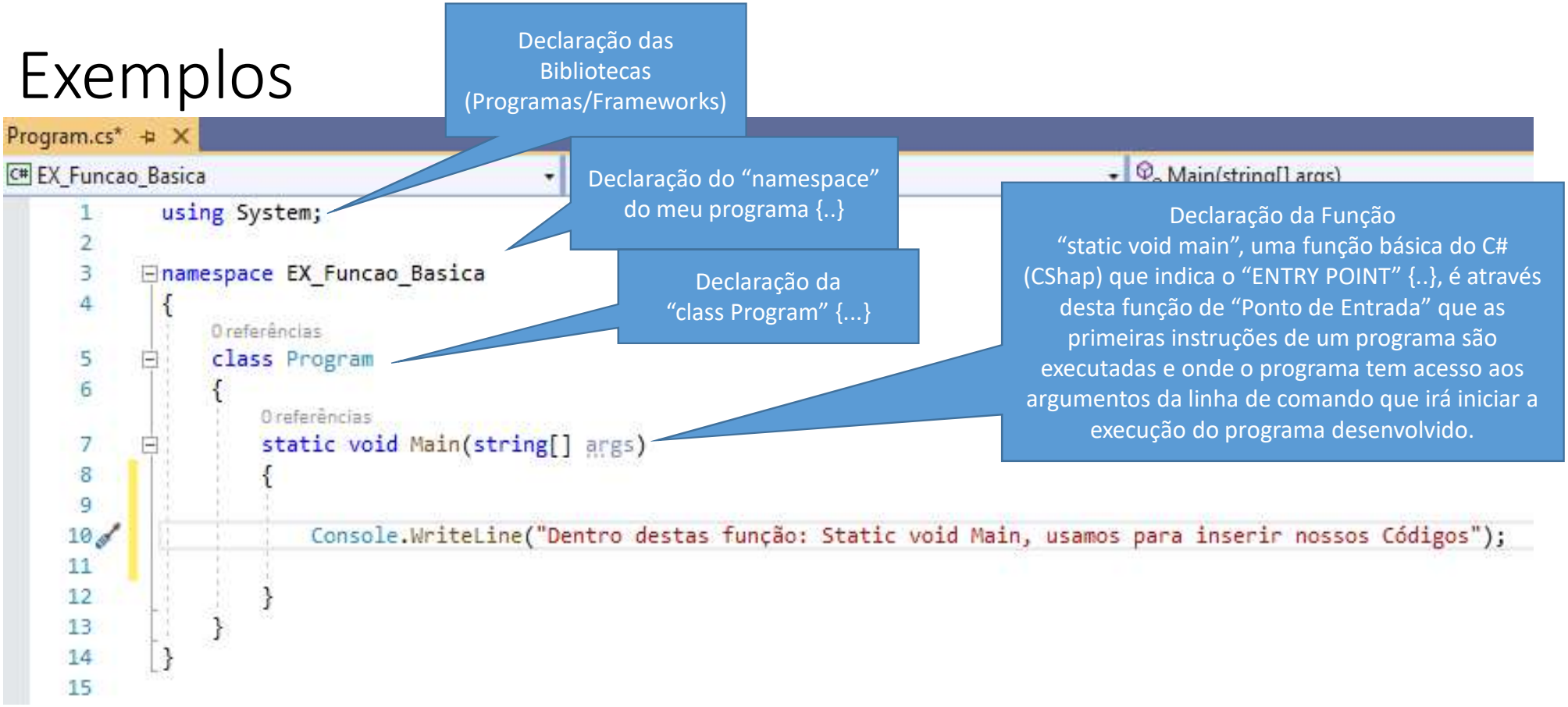
Exemplo: Math.Sqrt, é uma função criada e já esta incorporadas aos frameworks que usamos na construção dos programas em C#.

Modularização (Dividir seu programa em partes, sendo que estas partes eu vá reutilizar em diversas outras etapas do meu programa;

Delegação (Dirigindo as chamadas conforme a lógica exigida, ficando mais simples ao entendimento da rotina, uma forma mais limpa)

Reaproveitamento (Simplesmente a reutilização da mesma quando for necessário)

Exemplos



Toda a vez que iniciamos um projetos de programação em C# (Console.Net), nossa área de edição se apresenta da forma acima representada, isto para facilitar o programador em seu desenvolvimento, agara analise as definições informadas...

UdW1

Slide 3

UdW1 Usuário do Windows; 31/03/2021

Criando uma Função (EX_Funcao_Basica)

```
static void Main(string[] args)
{
        double n1, n2;

        Console.WriteLine("Usuário Informe dois (2) Números:");
        Console.Write("Informe o 1º Numero:");
        n1 = double.Parse(Console.ReadLine());
        Console.Write("Informe o 2º Numero:");
        n2 = double.Parse(Console.ReadLine());

        Console.WriteLine(soma(n1, n2).ToString("F2"));

        Console.WriteLine("Pressionae [ENTER] para encerrar o Console...");
        Console.ReadLine();

}
```

```
static double soma(double a, double b)
{
        double s;
        s = a + b;
        return s;
}
```

A variável “s” como vai responsável pelo “return” para a função, tem que ser do mesmo tipo da função “soma”.

Os métodos **static** ou métodos da classe são funções que não dependem de nenhuma variável de instância, quando invocados executam uma função sem a dependência do conteúdo de um objeto ou a execução da instância de uma classe

- Observar sempre onde estamos desenvolvendo nossa lógica de programação e suas variáveis, ou seja, qual o “scopo” ou local/método/área/região... (da aplicação/programa).

Codificação (EX_Funcao_Basica)

```
using System;
namespace EX_Funcao_Basica
{
    class Program
    {

   // Conteúdo ao Lado – Observar

    }
}
```

```
//Conteúdo  do “class programa”
 static void Main(string[] args)
 {
        double n1, n2;
        Console.WriteLine("Usuário Informe dois (2) Números:");
        Console.Write("Informe o 1º Numero:");
        n1 = double.Parse(Console.ReadLine());
        Console.Write("Informe o 2º Numero:");
        n2 = double.Parse(Console.ReadLine());
        Console.WriteLine(soma(n1, n2).ToString("F2"));
        Console.WriteLine("Pressionae [ENTER] para encerrar o Console...");
        Console.ReadLine();
}
static double soma(double a, double b)
{
       double s;
        s = a + b;
        return s;
}
```

DEBUG com Visual Studio 2019

Tópicos

- Teclas
  - F9 - marcar/desmarcar breakpoint
  - F5 - iniciar/continuar o debug
  - F10 - executar um passo (pula função)
  - F11 - executar um passo (entra na função)
  - SHIFT+F11 - sair do método em execução
  - SHIFT+F5 - parar debug

- Janelas
  - Watch (expressões personalizadas)
  - Autos (expressões "interessantes" detectadas pelo Visual Studio)
  - Locals (variáveis locais)

Introdução a OOP (Object-Oriented Programming)

Programação Orientada a Objetos (POO) é um modelo de análise, projeto e programação de software baseado na composição e interação entre diversas unidades chamadas de “Objetos”‘, esta programação é um dos quatro principais paradigmas de programação (+ programação imperativa, funcional e lógica). Os objetos são operados com o conceito de ”this” (isto) ou “self” (auto/si mesmo), de forma que seus métodos (muitas vezes) modifiquem os dados da própria instância. Os programas são arquitetados através de objetos que interagem entre si. Dentre as várias abordagens da POO, as baseadas em classes são as mais comuns: objetos são instâncias de classes, o que em geral também define o tipo do objeto. Cada classe determina o comportamento (definido nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos. A alternativa mais comum a utilização de classes é o uso de protótipos. Neste caso, objetos são cópias de outros objetos, não instâncias de classes. A diferença prática mais evidente é que na POO baseada em protótipos apenas a herança simples é implementada pela cópia do objeto.

Resumindo: OOP

Orientação a Objeto é um conceito que esta relacionado com a ideia de classificar, organizar e abstrair coisas. Veja a definição formal:

"O termo orientação a objetos significa organizar o mundo real como uma coleção de objetos que incorporam estrutura de dados e um conjunto de operações que manipulam estes dados."

"A programação Orientada a Objetos é um modelo de programação onde diversas classes possuem características que definem um objeto (na vida real) ... São exemplos de linguagens de programação orientadas a objetos: C++, Java, C#, Object Pascal, entre outras."

Programação Simples

Problema exemplo

Fazer um programa para ler as medidas dos lados de dois triângulos X e Y (suponha medidas válidas). Em seguida, mostrar o valor das áreas dos dois triângulos e dizer qual dos dois triângulos possui a maior área.

A fórmula para calcular a área de um triângulo a partir das medidas de seus lados a, b e c é a seguinte (fórmula de Heron):

$$area = \sqrt{p(p-a)(p-b)(p-c)} \qquad \text{onde} \qquad p = \frac{a+b+c}{2}$$

Exemplo:

```
Entre com as medidas do triângulo X:
3.00
4.00
5.00
Entre com as medidas do triângulo Y:
7.50
4.50
4.02
Área de X = 6.0000
Área de Y = 7.5638
Maior área: Y
```

Codificação Simples: (EX_AreaTriangulo_Basica)

```
//Declarando Variáveis
double p, areaX, areaY;
double xA, xB, xC, yA, yB, yC;
//Entrada de Dados de X
Console.WriteLine("Entre com as medidas do triângulo X:");
xA = double.Parse(Console.ReadLine(), CultureInfo.InvariantCulture);
xB = double.Parse(Console.ReadLine(), CultureInfo.InvariantCulture);
xC = double.Parse(Console.ReadLine(), CultureInfo.InvariantCulture);
//Entrada de Dados de Y
Console.WriteLine("Entre com as medidas do triângulo Y:");
yA = double.Parse(Console.ReadLine(), CultureInfo.InvariantCulture);
yB = double.Parse(Console.ReadLine(), CultureInfo.InvariantCulture);
yC = double.Parse(Console.ReadLine(), CultureInfo.InvariantCulture);
```

```
//Calculo Area X
p = (xA + xB + xC) / 2.0;
areaX = Math.Sqrt(p * (p - xA) * (p - xB) * (p - xC));
//Calculo Area Y
p = (yA + yB + yC) / 2.0;
areaY = Math.Sqrt(p * (p - yA) * (p - yB) * (p - yC));
//Apresentando Areas
Console.WriteLine("Área de X = " + areaX.ToString("F4", CultureInfo.InvariantCulture));
Console.WriteLine("Área de Y = " + areaY.ToString("F4", CultureInfo.InvariantCulture));

//Consistencia de Maior ou Igualdade
if (areaX > areaY)
{
   Console.WriteLine("Maior área: X");
} else if (areaX == areaY)
   {
       Console.WriteLine("Áreas Iguais entre X e Y");
   }
else
{
   Console.WriteLine("Maior área: Y");
}
```

C# - Programação Orientado a Objetos

Vamos Iniciar os conceitos para OOP / POO, trabalhando:

Classes, Atributos, Métodos e Membros Estáticos.

Para darmos continuação ao nosso Problema exemplo anterior, vamos criar uma classe com 3 atributos (relacionado a nossa entidade que é o Triangulo)

Entidade: Podemos definir como uma representação de um conjunto de informações sobre determinado conceito do sistema.

Ex: Cliente, Produto, Fornecedor, etc.

Classe

- É um tipo estruturado que pode conter (membros):
  - Atributos (dados / campos)
  - Métodos (funções / operações)
- A classe também pode prover muitos outros recursos, tais como:
  - Construtores
  - Sobrecarga
  - Encapsulamento
  - Herança
  - Polimorfismo
- Exemplos:
  - Entidades: Produto, Cliente, Triangulo
  - Serviços: ProdutoService, ClienteService, EmailService, StorageService
  - Controladores: ProdutoController, ClienteController
  - Utilitários: Calculadora, Compactador
  - Outros (views, repositórios, gerenciadores, etc.)

Discussão

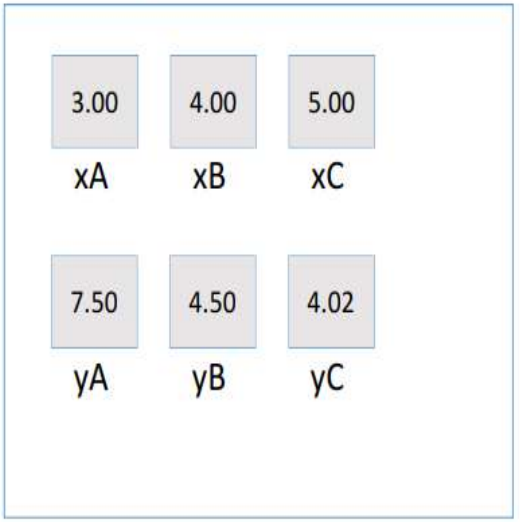
Triângulo é uma entidade com três atributos: a, b, c.

Estamos usando três variáveis distintas para representar cada triângulo:

```
double xA, xB, xC, yA, yB, yC;
```

Para melhorar isso, vamos usar uma CLASSE para representar um triângulo.

Memória:



Classe: Podemos definir como uma descrição que abstrai um conjunto de objetos com características similares.

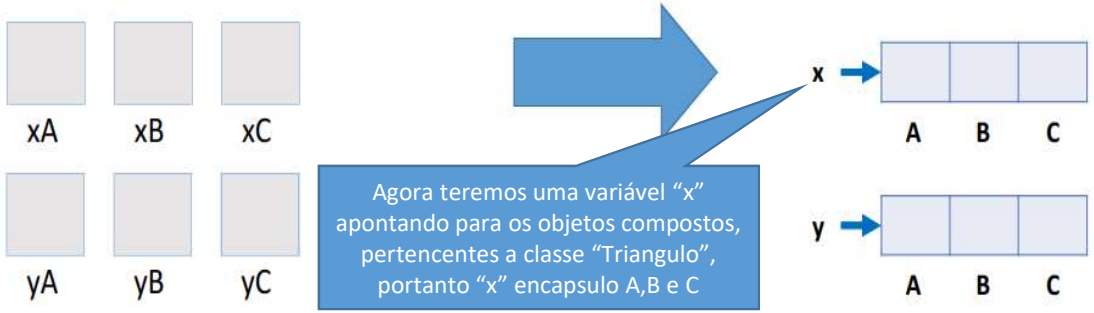
```
namespace Course {
    class Triangulo {

        public double A;
        public double B;
        public double C;
    }
}
```

Codificação:
Criando um Classe Triangulo .
Obs.: Como uma boa prática de construção, sempre iniciar uma classe com letra maiúscula.

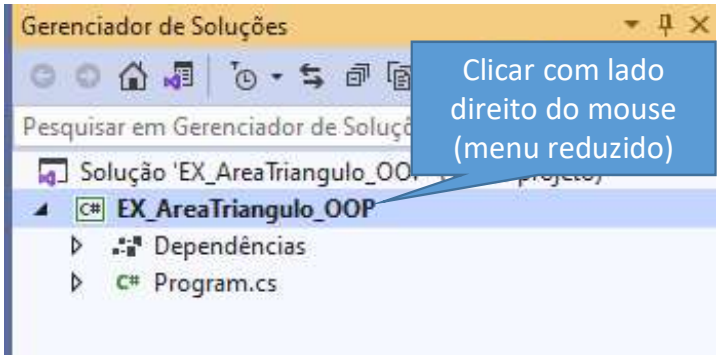
Para utilizar nossa classe "Triangulo", devemos instanciar nossas variáveis x e y

```
Triangulo x, y;
x = new Triangulo();
y = new Triangulo();
```

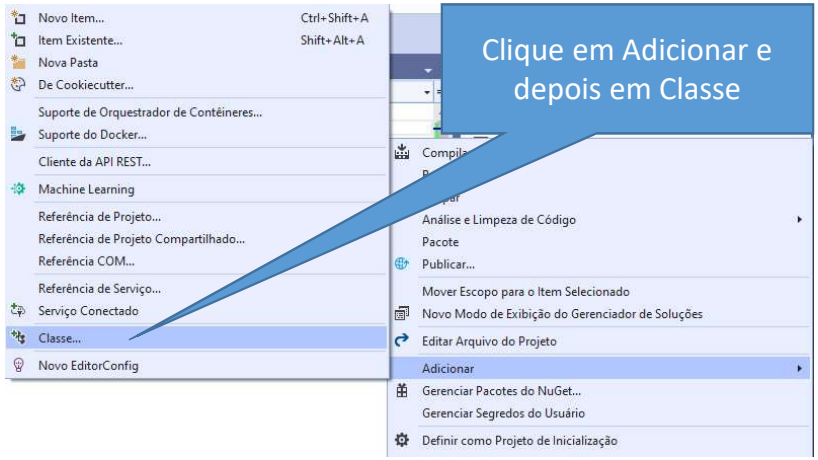
```
double xA, xB, xC, yA, yB, yC;
```



Agora teremos uma variável "x" apontando para os objetos compostos, pertencentes a classe "Triangulo", portanto "x" encapsulo A,B e C

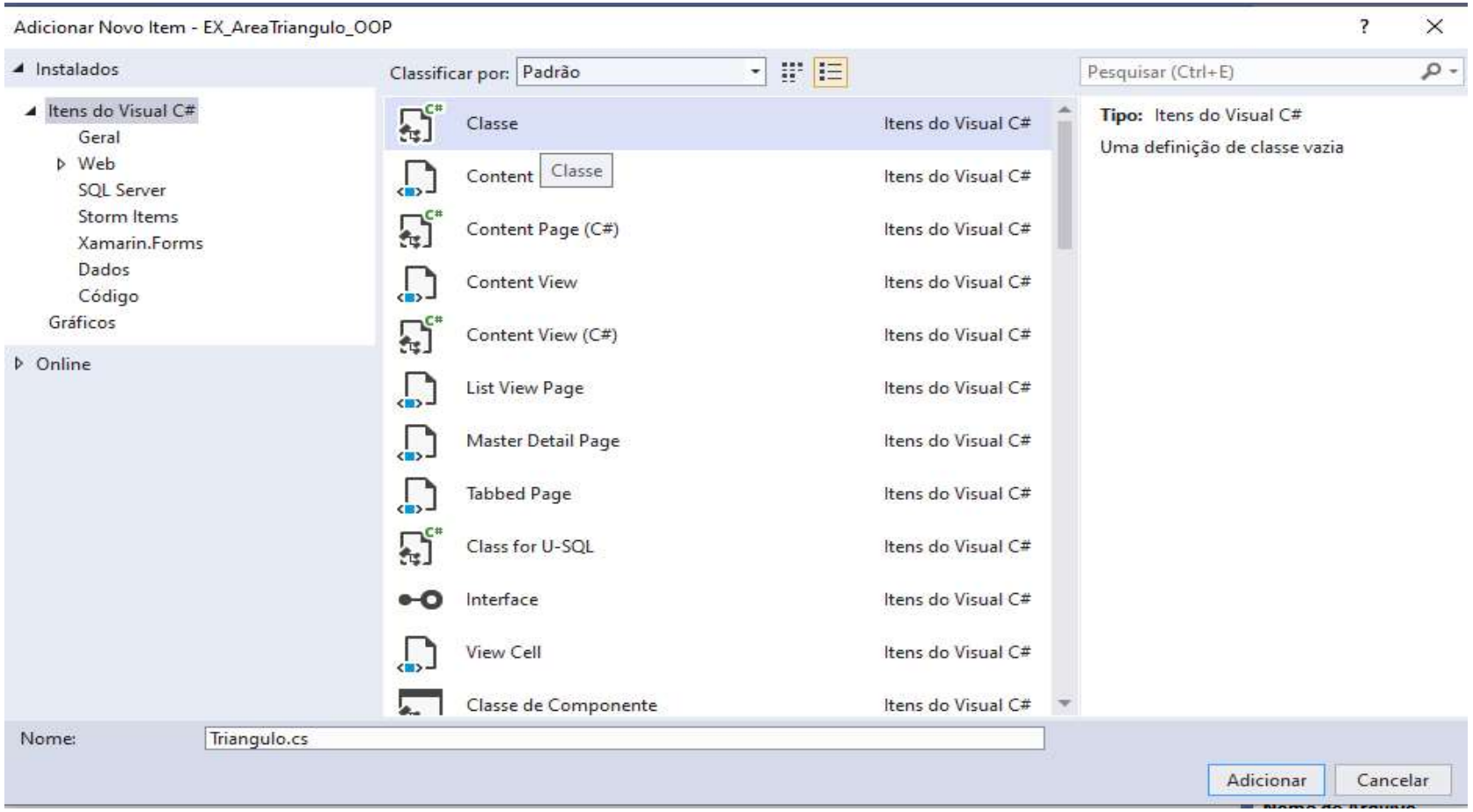


Clicar com lado direito do mouse (menu reduzido)



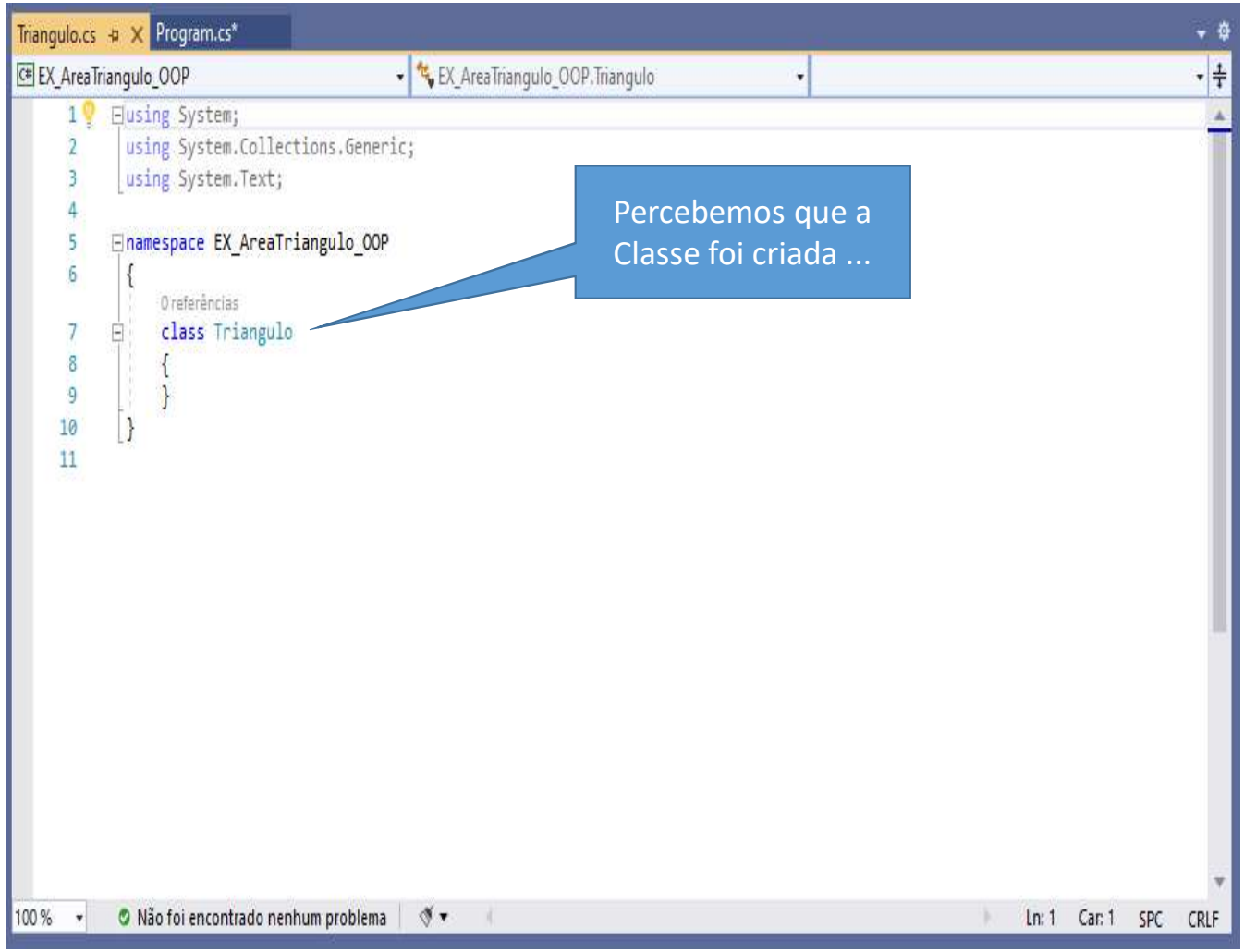
Clique em Adicionar e depois em Classe

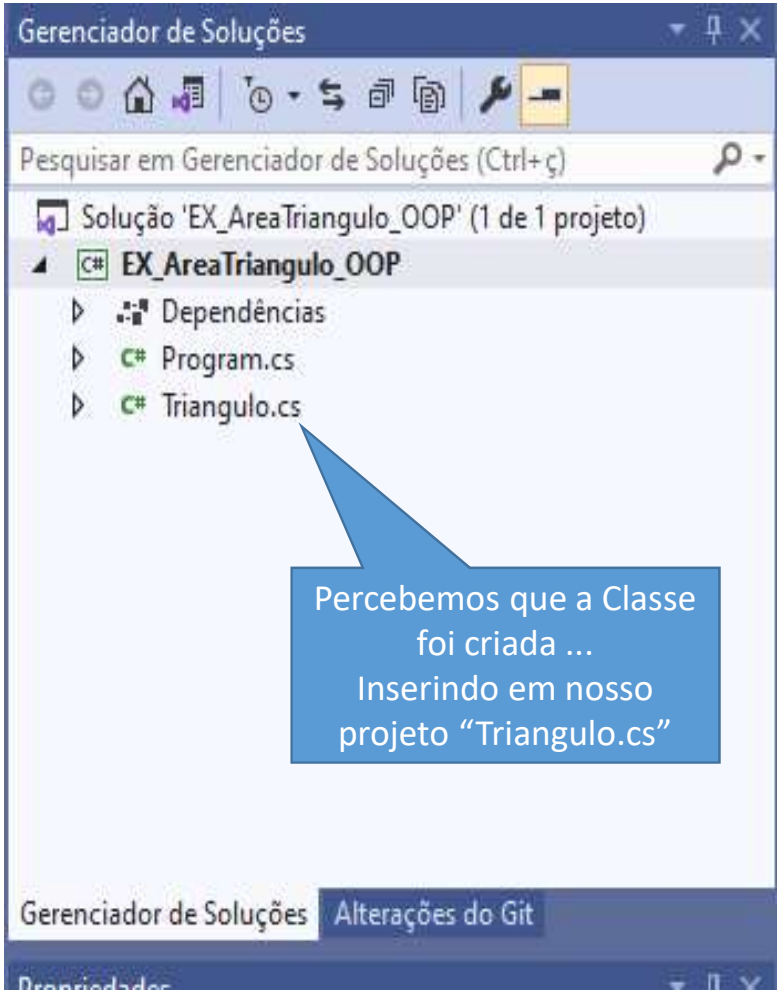
Adicionar Novo Item ...



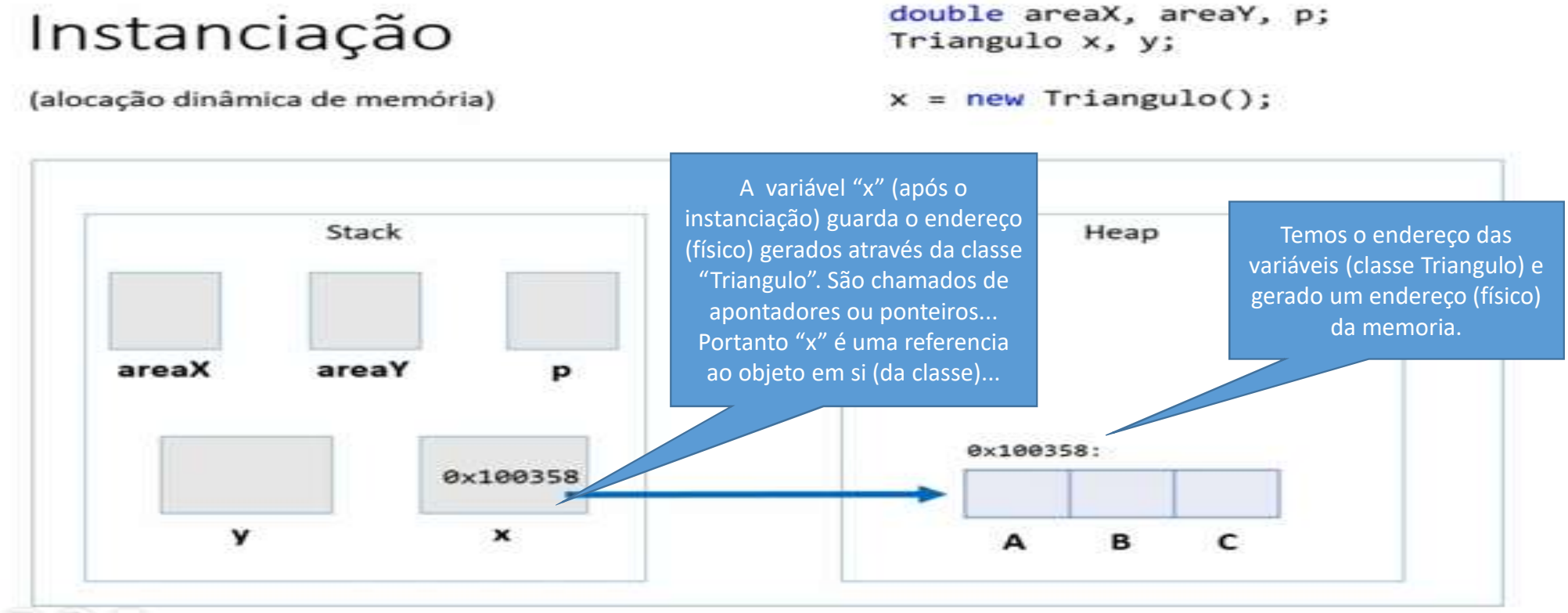
Certifique-se que esta selecionado o Item “Classe”, e insira o Nome, de preferencia fazendo referencia a entidade para o qual estamos criando a Classe. No nosso caso “Triangulo”...

Na Edição do nosso Código...





Como a Memoria (RAM) se Comporta:



Stack: "Pilha" é uma área específica da memória RAM usada para armazenar variáveis temporárias (estáticas) durante a execução do programa.

Heap: "Montoar", é uma área (RAM) de alocação dinâmica de variáveis, por exemplo, se um programa utiliza uma lista encadeada, este aloca essa estrutura que cresce dinamicamente.

Ponteiros ou Apontadores: Ponteiros "variáveis" que guardam o endereço de memória, para localização física armazenada na memoria.

Vamos ao Código da Classe “Triangulo” (EX_AreaTriangulo_OOP)

```
class Triangulo {
        public double A, B, C;
}
```

Ou

```
class Triangulo {
        public double A;
        public double B;
        public double C;
}
```

Código do Programa (EX_AreaTriangulo_OOP) – 1ª Parte

```
//Declaramos as variáveis "x" e "y" como do tipo "Triangulo" de acordo com a classe criada
Triangulo x, y;
//Declarando as variáveis "p", "areaX" e "areaY" do tipo double
double p, areaX, areaY;
//Instanciar as variáveis
x = new Triangulo();
y = new Triangulo();
/*
  * Neste programa estamos usando a inserção dos números com casas decimais separadas por "."
  * Portanto, inserimos "System.Globalization" através do comando "using" no inicio do nosso código
  * Com isso podemos usar o método "CultureInfo.InvariantCulture"...
*/
```

Código do Programa (EX_AreaTriangulo_OOP) – 2ª Parte

```
//Entrada de Dados de X, com seus atributos/parâmetros (classe)
Console.WriteLine("Entre com as medidas do triângulo X:");
x.A = double.Parse(Console.ReadLine(), CultureInfo.InvariantCulture);
x.B = double.Parse(Console.ReadLine(), CultureInfo.InvariantCulture);
x.C = double.Parse(Console.ReadLine(), CultureInfo.InvariantCulture);
//Entrada de Dados de Y, com seus atributos/parâmetros (classe)
Console.WriteLine("Entre com as medidas do triângulo Y:");
y.A = double.Parse(Console.ReadLine(), CultureInfo.InvariantCulture);
y.B = double.Parse(Console.ReadLine(), CultureInfo.InvariantCulture);
y.C = double.Parse(Console.ReadLine(), CultureInfo.InvariantCulture);
```

Código do Programa (EX_AreaTriangulo_OOP) – 3ª Parte

```
 //Calculo Area X
p = (x.A + x.B + x.C) / 2.0;
areaX = Math.Sqrt(p * (p - x.A) * (p - x.B) * (p - x.C));
//Calculo Area Y
p = (y.A + y.B + y.C) / 2.0;
areaY = Math.Sqrt(p * (p - y.A) * (p - y.B) * (p - y.C));
//Apresentando Areas
Console.WriteLine("Área de X = " + areaX.ToString("F4", CultureInfo.InvariantCulture));
Console.WriteLine("Área de Y = " + areaY.ToString("F4", CultureInfo.InvariantCulture));
//Consistencia de Maior ou Igualdade
if (areaX > areaY) {
   Console.WriteLine("Maior área: X");
} else if (areaX == areaY)  {
            Console.WriteLine("Áreas Iguais entre X e Y");
} else {
          Console.WriteLine("Maior área: Y");
}
          Console.ReadLine();
```

Vários Testes e seus resultados

```
Entre com as medidas do triângulo X:
3.00
4.00
5.00
Entre com as medidas do triângulo Y:
7.50
4.50
4.02
Área de X = 6.0000
Área de Y = 7.5638
Maior área: Y
```

Áreas Iguais

```
Entre com as medidas do triângulo X:
9
8
7
Entre com as medidas do triângulo Y:
7
8
9
Área de X = 26.8328
Área de Y = 26.8328
Áreas Iguais entre X e Y
```

Ignora a virgula, ou seja, 5,5 = 55

```
Entre com as medidas do triângulo X:
5.5
4.5
3.5
Entre com as medidas do triângulo Y:
5,5
4,5
3,5
Área de X = 7.8549
Área de Y = 785.4885
Maior área: Y
```

Não Calcula, pois não tem com achar área de um triangulo que não existe.

```
Entre com as medidas do triângulo X:
8
4
2
Entre com as medidas do triângulo Y:
10
5
1
Área de X = NaN
Área de Y = NaN
Maior área: Y
```

Podemos testar usando uma pagina (web):

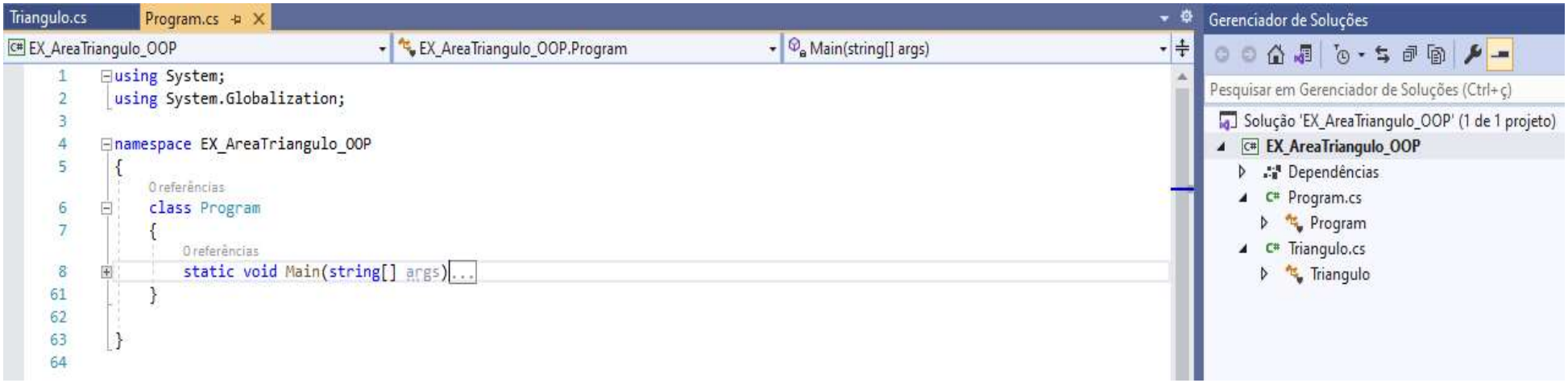
https://www.calkoo.com/pt/calculadora-do-triangulo

Comparações e Conclusões

Analisar o Código (Ex_Triangulo_Classe):

Observe que a classe foi criada dentro do program.cs (namespace) que utiliza seus objetos (referencias)...

Código (EX_AreaTriangulo_OOP)



Reflexão

Classes, objetos, atributos

- Classe: é a definição do tipo

```
namespace Course {
    class Triangulo {

        public double A;
        public double B;
        public double C;
    }
}
```

- Objetos: são instâncias da classe

- **Classe** descreve (agrupa) todos os **objetos** de um tipo particular;
- **Objetos** possuem características próprias, denotadas por **atributos**.

Exercícios (Desafios)

Criar Classe; Criar Objetos da Classe (Objetos com o Tipo sendo a Classe criada) para utilização no Programa "Principal".

1) Fazer um programa para ler os dados de duas pessoas, depois mostrar o nome da pessoa mais velha.
2) Fazer um programa para ler nome e salário de dois funcionários. Depois, mostrar o salário médio dos funcionários, com o “separador” de casas decimais no padrão mundial, ou seja, trocar “,” por “.”, mãos a obra.

LP_Exercicicios_Slides_06