

Escola Técnica Estadual “São Paulo” - ETESP

Linguagem C# - Visual Studio 2019

PROGRAMAÇÃO ORIENTADA A OBJETOS

Os caminhos da programação têm mudado desde a invenção do computador. Quando os computadores foram inventados, a programação era feita por chaveamentos em instruções binárias de máquina, usando-se o painel frontal. Enquanto os programas continham somente algumas centenas de instruções, esse método funcionava.

A linguagem assembly foi desenvolvida para permitir a um programador manipular “programas mais complexos”. A primeira linguagem de alto nível difundida foi o FORTRAN. Ainda que o FORTRAN tenha dado um primeiro passo bastante considerável, é uma linguagem que somente torna os programas mais claros e fáceis de entender do que o assembly, sem introduzir mudanças consideráveis no estilo de programação.

Nos anos 60 nasceu a programação estruturada. Esse é o método utilizado por linguagens como C e Pascal. Foi possível, pela primeira vez, escrever programas complexos de maneira razoavelmente fácil.

A programação orientada a objetos aproveitou as melhores ideias da programação estruturada, permitindo que um problema seja mais facilmente decomposto em subgrupos relacionados. Então, usando-se a linguagem, pode-se traduzir esses subgrupos em objetos.

Mas afinal o que é um objeto?

De forma genérica “um objeto é a descrição de qualquer coisa que exista”.

Os objetos podem ser:

- Físicos: Um carro, uma casa, uma pessoa, um cachorro ...
- Conceituais: Um organograma, uma ideia ...
- Uma entidade: Um botão, um ComboBox, uma planilha...

Quando restringimos ao mundo da programação podemos dizer que um objeto é algo que tem características (propriedades) e ações (métodos).

Os termos “características e ações” são traduzidos para a Orientação a Objeto (OO) da seguinte forma:

- Características: São as **Propriedades**. Ex: Nome, Tipo, Cor, Tamanho, Quantidade...
- Ações: O que o objeto sabe fazer, por exemplos: Andar, Falar, Comer, Quebrar, Salvar.

Vantagens da Programação Orientada a Objetos:

- A capacidade de criar sistemas de software através de um processo de equipe, permitindo a especialistas trabalharem em partes do sistema;
- A capacidade de reutilização de componentes de código em outros programas e a aquisição de componentes escritos por desenvolvedores de terceiros para aumentar a funcionalidade dos seus programas com pouco esforço.

Conceitos básicos utilizados na “Programação Orientada a Objetos”

- **Abstração:** Quando interagimos com objetos no “mundo real”, não estamos preocupados com os “detalhes sobre o funcionamento destes objetos”. Quando decidimos ouvir música, simplesmente ligamos o aparelho e ouvimos a música (não nos preocupamos em como a música chega aos nossos ouvidos). Sem esta capacidade de abstrair ou filtrar as propriedades de objetos teremos muita dificuldade para processar o excesso de informação que nos bombardeia diariamente.

- **Classe:** Descrição (modelo - molde) de um objeto. Para que o objeto exista, deve existir a classe. Normalmente dizemos: A classe “gera” o objeto. O objeto existe a partir da classe. A classe pode gerar muitas instâncias (cópias) do objeto. **Por convenção, os nomes das classes em C# devem iniciar com o primeiro caractere em “maiúsculo”.**

- **Instância:** Seria como dizer: dar vida a uma classe, a um modelo (estamos CRIANDO o objeto).

- **Construtor:** É o método “especial” que será SEMPRE executado quando a classe for “instanciada” (criada). Normalmente utilizado para inicialização das variáveis, abertura do Banco de Dados. É um “método especial” pois **DEVERÁ ter o mesmo nome da classe e não retorna valores (e também NÃO DECLARAMOS a palavra VOID).**

- **Objeto:** Um objeto é então uma instância de uma classe.

- **Encapsulamento:** Processo no qual o acesso direto aos dados de um objeto não é permitido, em vez disso, ele está escondido. O encapsulamento dos dados o torna mais seguros e confiáveis, sabemos como os dados estão sendo acessados e quais operações estão sendo realizadas sobre os mesmos.

- **Herança:** Pode ser definida como a capacidade de uma classe “herdar” atributos (propriedades) e comportamentos (métodos) de uma outra classe. É utilizado quando necessitamos que uma determinada classe tenha todas as características de outra classe já existente, porém, com algumas modificações em seu comportamento, ou mesmo algumas funcionalidades adicionais.

- **Generalização e Especialização:** As palavras **generalização e especialização** fazem parte do conceito de Herança. Uma classe “mais genérica” (generalização) pode gerar uma nova classe que herda suas características, mas permitindo modificações e novas implementações, ou seja, a nova classe é uma especialização da classe genérica.

Um exemplo tradicional usado para explicar o conceito de herança é a definição das classes Pessoa, PessoaFisica e PessoaJuridica. A **classe Pessoa é genérica**, contém atributos, métodos e propriedades que podem representar qualquer pessoa. Já as **classes PessoaFisica e PessoaJuridica** representam, respectivamente, as características para uma pessoa física e pessoa jurídica e **são classes especializadas**, pois especializam a classe Pessoa.

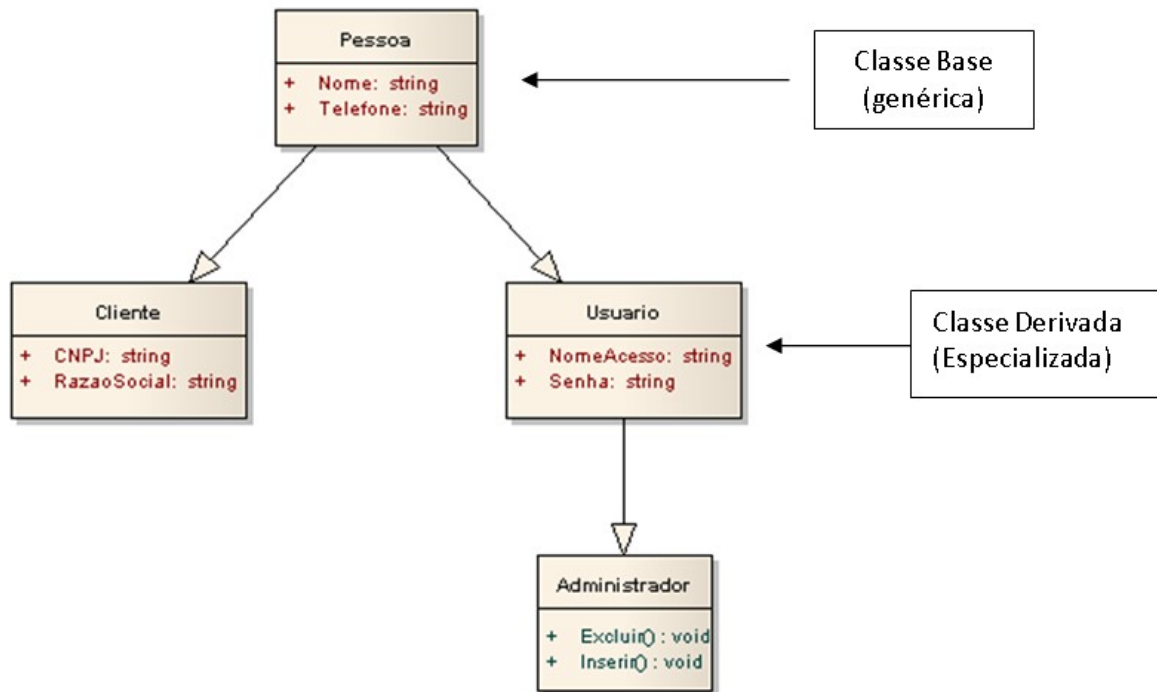
A classe cujos membros são herdados (**classe genérica**) é chamada de **classe base** e a classe que herda esses membros (**classe especializada**) é chamada de **classe derivada**.

Conceitualmente, uma classe derivada é uma especialização da classe base.

Quando você derivar, implicitamente obterá todos os membros da classe base. A classe derivada, assim, pode reutilizar o código na classe base sem ter que reimplementar a ele. Na classe derivada, você pode adicionar mais membros. Dessa forma, a classe derivada estende a funcionalidade da classe base.

Em C# uma relação de Herança é implementada com o uso do símbolo “:” (dois pontos) na definição da classe.

Observe a figura:



- **Polimorfismo:** Literalmente, polimorfismo significa “muitas formas”. No mundo da programação, polimorfismo é muitas vezes definido como a “capacidade de objetos diferentes possuírem métodos de mesmo nome e mesma lista de parâmetros que quando chamados executam tarefas de maneiras diferentes”.

Etapas para construção da classe - Atividade prática:

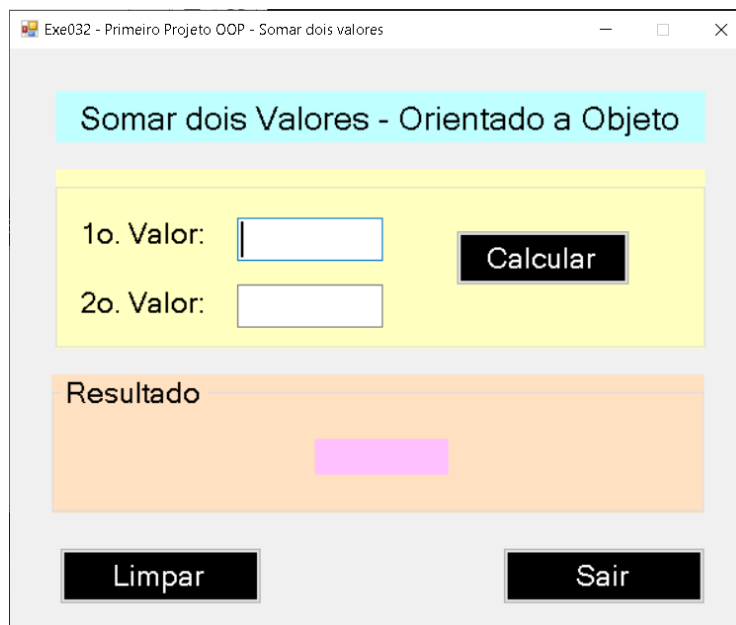
Estas definições (e outras que serão apresentadas posteriormente), sem exemplos práticos, não ajudam o desenvolvedor que está tentando desmistificar o paradigma da POO, por isso, faremos uma pausa na teoria...

Veremos alguns exemplos práticos dos conceitos apresentados até o momento.

Prof. Roberto de Castro

EXERCÍCIO PRÁTICO: A proposta do projeto consiste em receber dois valores (valor1 e valor2), efetuar a soma entre os dois e exibir o resultado.

Interface gráfica:



Normalmente representamos uma “classe” pelo diagrama UML abaixo:

ClsOperacao	← nome da classe
-valor1: int -valor2: int	← atributos da classe. OBS: O sinal “-” indica ser um atributo (campo) “privativo da classe” (visível somente dentro da classe).
+ Somar(valor1:int, valor2) int	← métodos da classe. O método “somar” recebe duas variáveis e retornar um valor. O sinal “+” indica que é um método público (visível fora da classe).

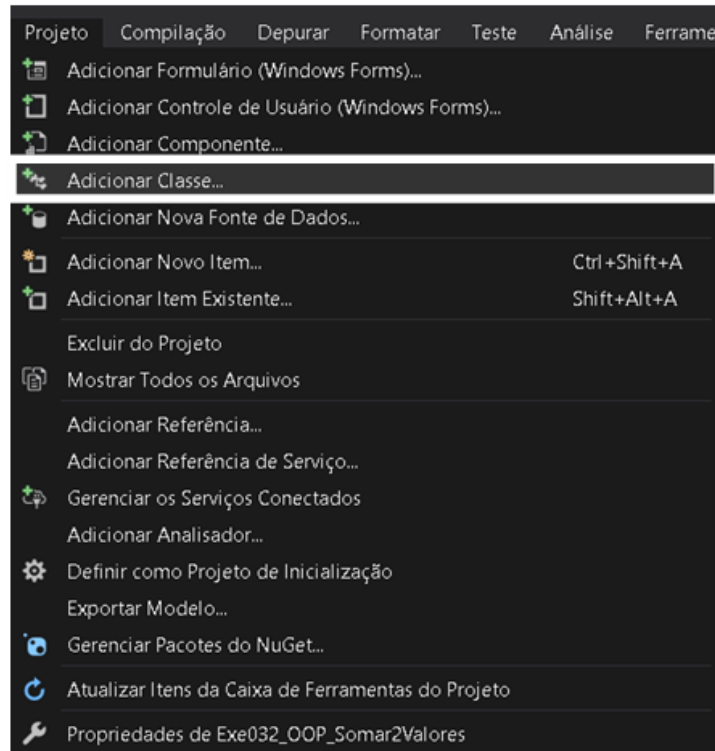
Principais etapas para a criação da classe:

- Declaração dos atributos/campo: Variáveis “privativas” da classe.
- Declaração dos métodos gets/sets: São as “propriedades visíveis ao mundo externo”. Funcionam como um “elo de ligação” entre as variáveis privativas da classe (atributos) e o programa “cliente”. São elas que recebem e enviam dados.
- Declaração do construtor: O construtor é um método “especial” utilizado para inicializar os objetos da classe quando estes são criados (instanciados). Este método possui o mesmo nome da Classe e não tem nenhum tipo de retorno, nem mesmo void.
- Declaração dos métodos (procedimentos): São as ações que nossos objetos podem executar.

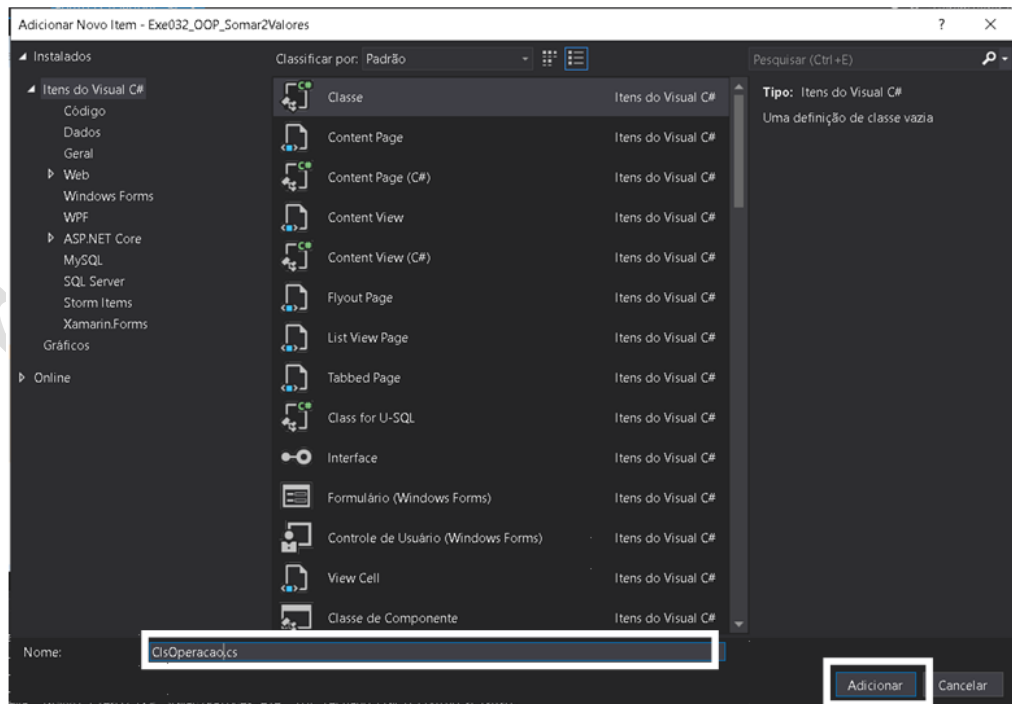
Prof. Roberto de Castro

Principais etapas, após o “desenho da interface gráfica” no programa principal:

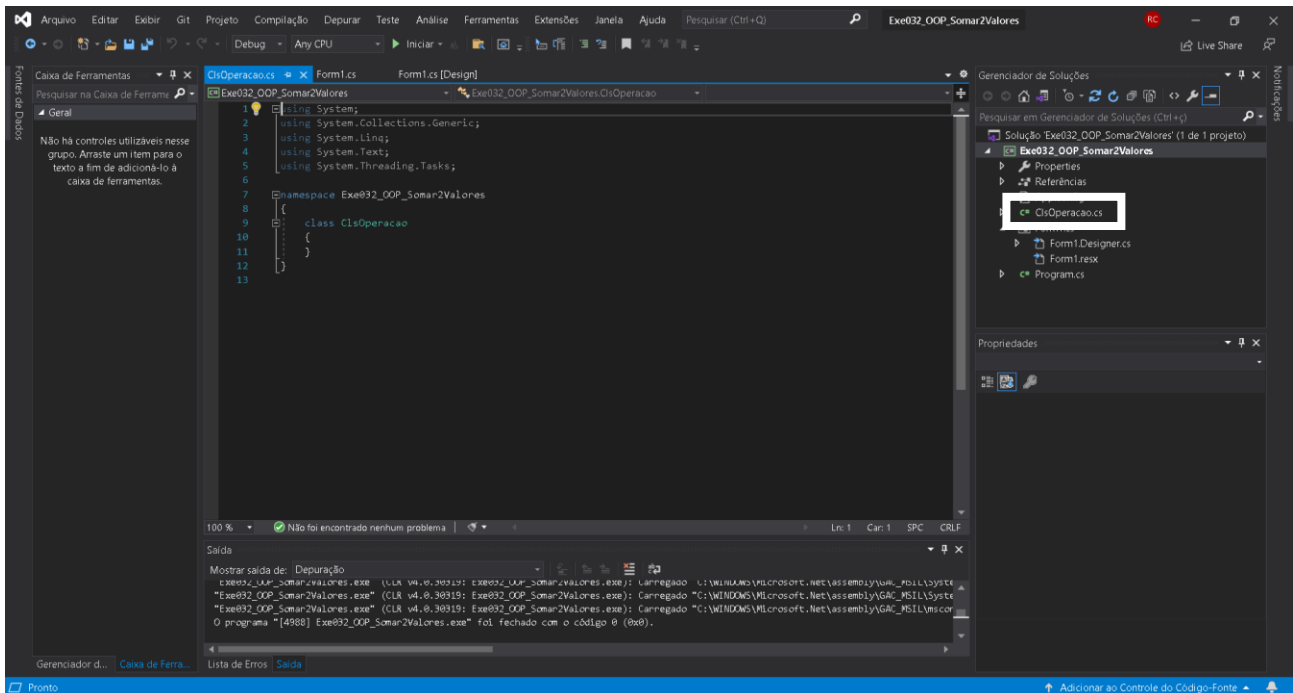
1. Adicionar a “classe” ao projeto. Clicar no menu “Project” (projeto) → Selecionar a opção Add Class (Adicionar classe). Veja o modelo:



2. Na próxima “tela”, digite o nome da classe. No nosso exemplo: ClsOperacao e clique em Adicionar



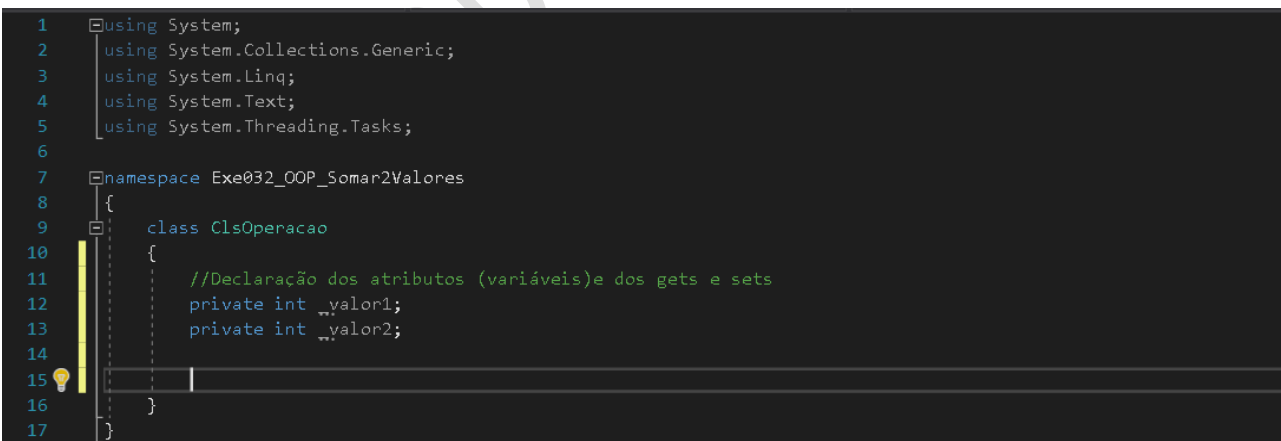
3. O “Visual Studio” exibe (abre) a área para programação da classe. Observe, na janela do “Solution Explorer” (lado direito do projeto), um “novo arquivo”, a “nossa classe ClsOperacao!!! Será nesta área que iremos incluir nossa programação.



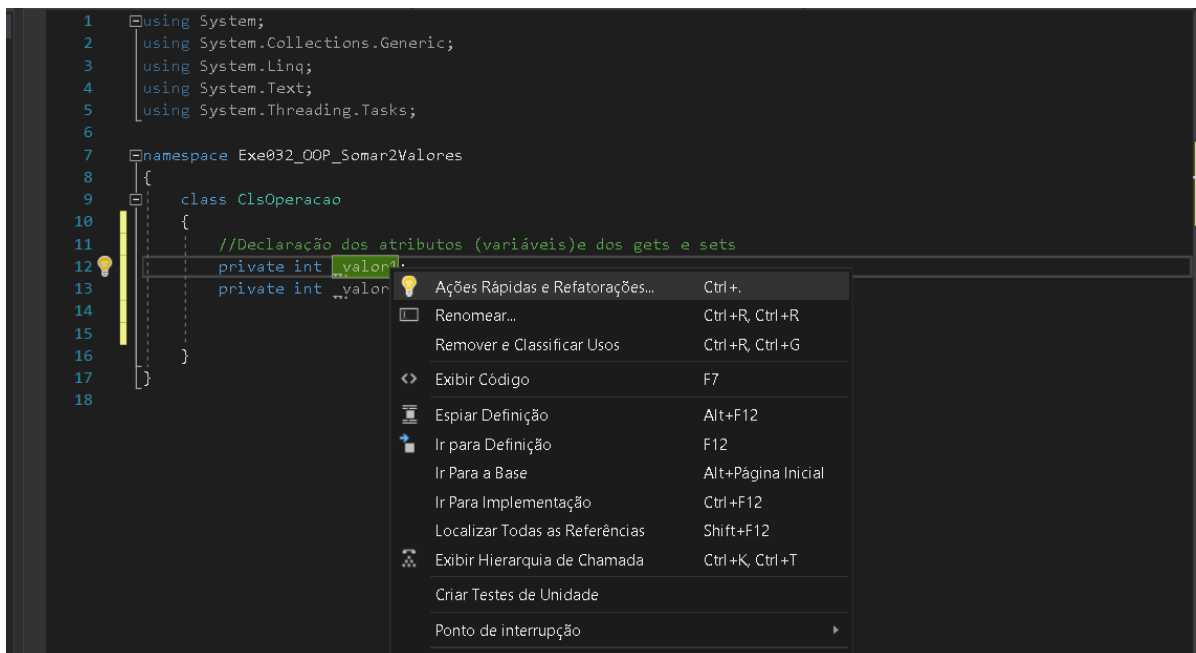
Programação da Classe “ClsOperacao”:

Seguiremos as etapas:

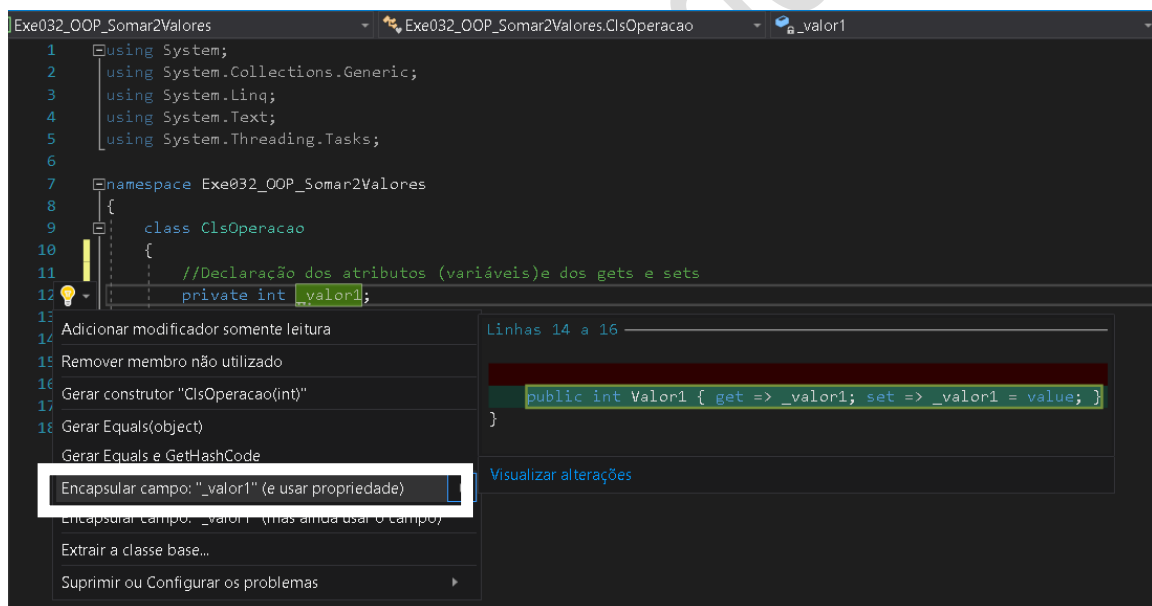
1. Declaração dos Campos/Atributos (variáveis privadas da classe). Neste exemplo serão duas variáveis inteiras: `_valor1` e `_valor2`;



2. Declaração das “propriedades” (Gets e Sets). Podemos entender como sendo as “variáveis públicas”, que servirão como “elo” de ligação/comunicação entre o programa principal e a classe. O programa principal irá fornecer dois valores que serão recebidos pela classe nestas propriedades e “transferidos” para as variáveis privadas (`_valor1` e `_valor2`). Este é o conceito de “encapsulamento” (proteção dos dados da classe). Estas propriedades serão criadas “automaticamente” pressionando-se o botão direito do mouse sobre a declaração dos atributos e selecionando a opção “Ações Rápidas e Refatorações”, veja a imagem:



Após clicar em “Ações Rápidas e Refatorações”, dê dois cliques em “Encapsular Campo”:



E o Visual Studio incluirá “automaticamente” a linha:

```
public int Valor1 { get => _valor1; set => _valor1 = value; }
```

Repita o processo acima, para criar a “segunda propriedade”. O Visual Studio deverá incluir a linha:

```
public int Valor2 { get => _valor2; set => _valor2 = value; }
```

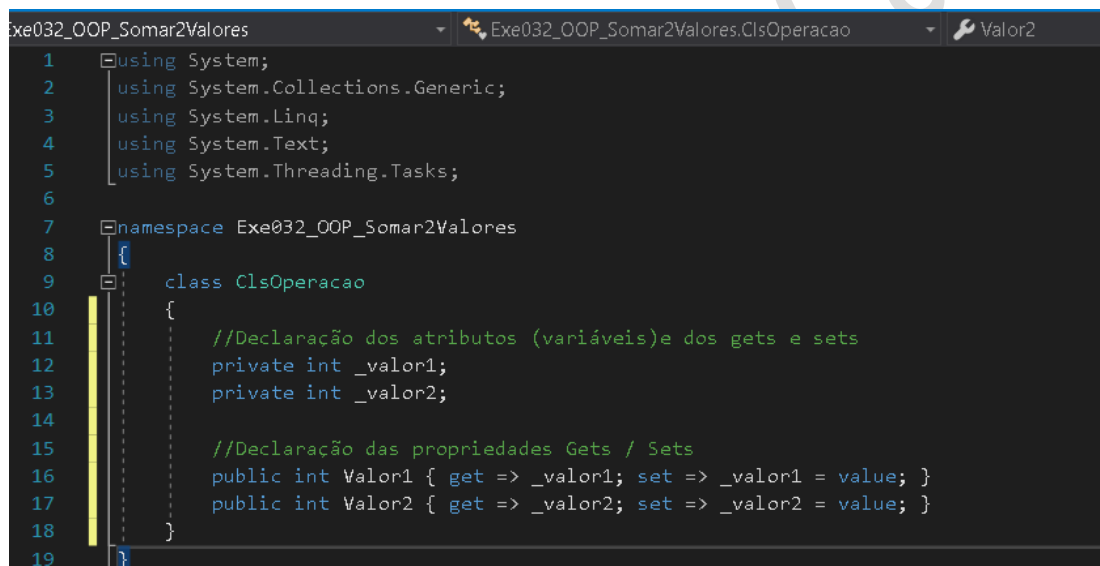
Podemos interpretar as declarações da seguinte forma: O programa principal irá transferir o “valor1” e o “valor2” para a classe “ClsOperacao” através da propriedade (“variável pública”) Valor1 e Valor2. A classe ao receber estes valores, transferirá (set) para as variáveis “privativas”: _valor1 e _valor2.

Veja, abaixo, como as propriedades “Gets/Sets” eram criadas pelo Visual Studio, em [versões anteriores](#):

```
public int Valor1
{
    get { return _valor1; }
    set { _valor1 = value; }
}
```

```
public int Valor2
{
    get { return _valor2; }
    set { _valor2 = value; }
}
```

Até o momento o projeto deverá estar com o formato abaixo:



```
Exe032_OOP_Somar2Valores
Exe032_OOP_Somar2Valores.ClsOperacao
Valor2

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Exe032_OOP_Somar2Valores
8  {
9      class ClsOperacao
10     {
11         //Declaração dos atributos (variáveis) e dos gets e sets
12         private int _valor1;
13         private int _valor2;
14
15         //Declaração das propriedades Gets / Sets
16         public int Valor1 { get => _valor1; set => _valor1 = value; }
17         public int Valor2 { get => _valor2; set => _valor2 = value; }
18     }
19 }
```

3. A terceira etapa é a declaração do método construtor. **Lembre-se:** trata-se de um “método especial”, que será executado SEMPRE que a classe for “instanciada” (executada), deverá ser “público”, ter o mesmo nome da classe e NÃO RETORNAR NENHUM VALOR (e também não declaramos como VOID).

Veja a imagem abaixo, linhas 17 a 22.


```
1
2 namespace Exe032_OOP_Somar2Valores
3 {
4     class ClsOperacao
5     {
6         //Declaração dos atributos (variáveis)e dos gets e sets
7         private int _valor1;
8         private int _valor2;
9
10        //Declaração das propriedades Gets / Sets
11        public int Valor1 { get => _valor1; set => _valor1 = value; }
12        public int Valor2 { get => _valor2; set => _valor2 = value; }
13
14
15        //Declaração do Método Construtor - este método possui o mesmo nome da classe e NÃO possui VOID
16        //Irà inicializar os atributos - variáveis privativas
17        public ClsOperacao()
18        {
19            _valor1 = 0;
20            _valor2 = 0;
21        }
22    }
23 }
```

4. A quarta etapa de criação da classe, é a declaração dos métodos (o que a classe irá fazer, qual será o processamento). Neste nosso projeto, o método efetuará a soma dos dois valores que foram recebidos na classe e devolverá o resultado da operação.

IMPORTANTE: Para que este método possa ser visualizado pelo programa principal, **DEVERÁ ser declarado como PUBLIC!!**

Resultado final, após a digitação do método para somar os valores:

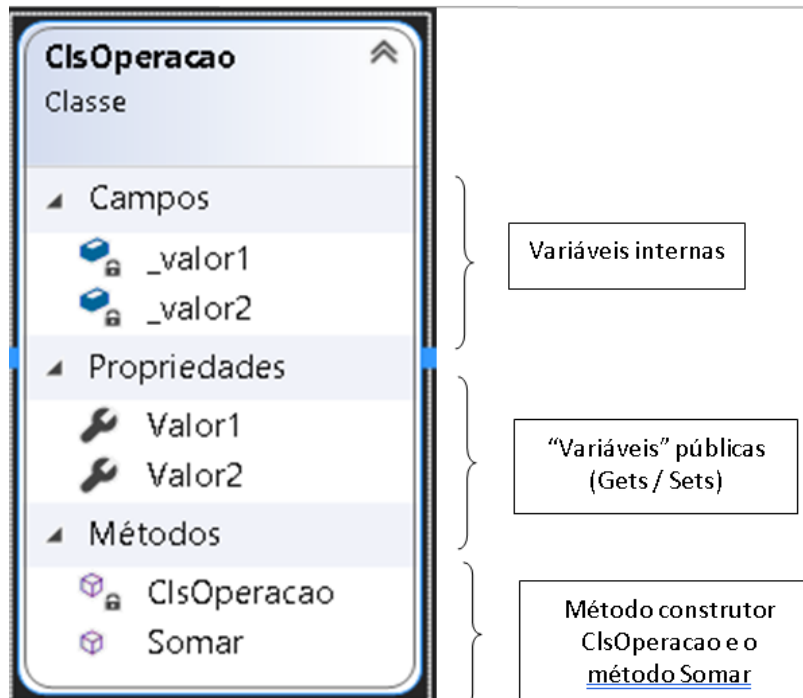
```
Exe032_OOP_Somar2Valores
Exe032_OOP_Somar2Valores.ClsOperacao
_valor1

2
3 namespace Exe032_OOP_Somar2Valores
4 {
5     class ClsOperacao
6     {
7         //Declaração dos atributos (variáveis)e dos gets e sets
8         private int _valor1;
9         private int _valor2;
10
11        //Declaração das propriedades Gets / Sets
12        public int Valor1 { get => _valor1; set => _valor1 = value; }
13        public int Valor2 { get => _valor2; set => _valor2 = value; }
14
15
16        //Declaração do Método Construtor - este método possui o mesmo nome da classe e NÃO possui VOID
17        //Irà inicializar os atributos - variáveis privativas
18        public ClsOperacao()
19        {
20            _valor1 = 0;
21            _valor2 = 0;
22        }
23
24        //Declaração do método que irá efetuar a soma entre os dois valores e devolver o resultado
25        public int Somar()
26        {
27            return _valor1 + _valor2;
28        }
29    }
30 }
```

Prof. Roberto de Castro

Finalmente nossa classe está pronta para ser executada, porém, ela sempre necessitará de um “programa principal”, que no nosso caso, é o projeto que contém a interface gráfica (formulário).

Podemos “visualizar” o diagrama da classe, pressionando o botão direito sobre o nome da classe, no Solution Explorer e selecionar a opção “Exibir em Diagrama de Classe”. O Visual Studio gerará o diagrama:



E agora a programação no programa principal...

```
using System;
using System.Windows.Forms;

namespace Exe032_OOP_Somar2Valores
{
    public partial class FrmExe032 : Form
    {
        public FrmExe032()
        {
            InitializeComponent();
        }

        private void BtnSair_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void BtnLimpar_Click(object sender, EventArgs e)
        {
            TxtValor1.Text = "";
            TxtValor2.Text = "";
            LblResultado.Text = "";
            TxtValor1.Focus();
        }
    }
}
```

```
private void BtnCalcular_Click(object sender, EventArgs e)
{
    //Gera a instância da classe. Neste momento o "construtor" é executado!!
    ClsOperacao ObjOperacao = new ClsOperacao();

    //Transfere para as variáveis da classe (propriedades/Get_Set), o valor dos "TextBox"
    ObjOperacao.Valor1 = Convert.ToInt32(TxtValor1.Text);
    ObjOperacao.Valor2 = Convert.ToInt32(TxtValor2.Text);

    int resultado = 0;

    //Executa o método Somar da classe ClsOperacao
    resultado = ObjOperacao.Somar();
    LblResultado.Text = resultado.ToString();
}
}
```

Considerações sobre a solução: Não implementamos os testes na "caixa de texto", com o objetivo de priorizar o entendimento da técnica da "programação orientada a objeto". **IMPORTANTE:** Caso não sejam digitados números nos "TextBox", o programa encerrará com uma mensagem de erro!

Atividade prática: Vamos reforçar os conceitos estudados até o momento... **EXE032B:** Vamos "revisitar" um projeto já resolvido (Consulta Lista Classificados no Vestibulinho) e "converter" para Programação Orientada a Objetos.

INTERFACE GRÁFICA

Exe032B - Consulta Classificados - POO

ESCOLA TÉCNICA ESTADUAL ETESP

CONSULTA RESULTADO VESTIBULINHO

NÚMERO DE INSCRIÇÃO: Consultar

Status:

Limpar Sair

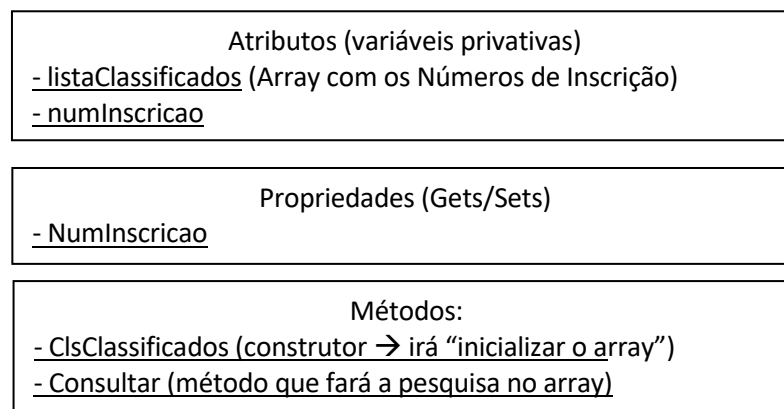
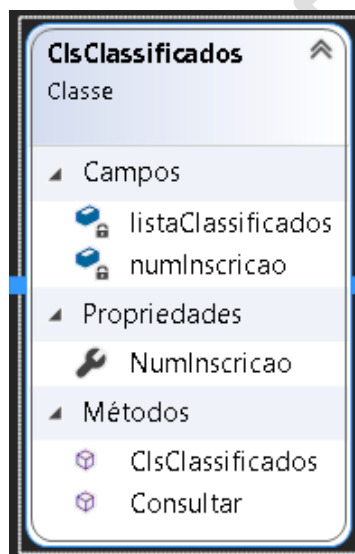
Prof. Roberto de Castro

LISTA DE CLASSIFICADOS	
Número de Ordem	(Número de Inscrição)
1	10514
2	30343
3	8240
4	3125
5	50525
6	23289
7	7310
8	9281
9	49524
10	33001

Considerações:

- Se o número de inscrição existir na “lista de classificados”, indica que o candidato foi classificado;
- A “posição” dentro da lista, indica qual é a “classificação” do candidato no vestibulinho..
- Se o número de inscrição NÃO existir na “lista de classificados”, exibir a mensagem “Lista de Espera”.
- Importante destacar que para a solução deste problema NÃO poderemos utilizar diretamente o “número de inscrição” como índice do array.

Diagrama de classe:



Principais etapas:

1. Desenho da interface gráfica
2. Criação e programação da classe "ClsClassificados":
 - a) Declaração dos atributos/campo: Variáveis "privativas" da classe.
 - b) Declaração dos métodos gets/sets: São as "propriedades visíveis ao mundo externo". Funcionam como um "elo de ligação" entre as variáveis privativas da classe (atributos) e o programa "cliente". São elas que recebem e enviam dados.
 - c) Declaração do construtor: O construtor é um método "especial" utilizado para inicializar os objetos da classe quando estes são criados (instanciados). Este método possui o mesmo nome da Classe e não tem nenhum tipo de retorno, nem mesmo void. Para este projeto, o "construtor" irá preencher o array com a lista dos classificados!
 - d) Declaração do método (procedimentos) Consultar → Fará uma pesquisa no array (Array.IndexOf) e irá retornar um texto, informando se o candidato está classificado ou não.
3. Programação do "botão" consultar (formulário principal).