

Material de Apoio complementar “propriedades” (Get/Set)

Antes de nos aprofundarmos em novos conceitos (sobrecarga, sobrescrita, herança...) vamos reforçar o conceito de “encapsulamento” e a utilização das “propriedades” (gets/sets).

Nos projetos desenvolvidos até o momento, declaramos as propriedades “gets/sets”, porém, nos exemplos apresentados, não utilizamos todas as potencialidades destes recursos, que tem como finalidade proporcionar o “encapsulamento” (um dos pilares da programação orientada a objetos, seguido de: abstração, polimorfismo e herança).

“O encapsulamento fornece uma maneira de preservar a integridade do estado dos dados. Ao invés de definir campos públicos devemos definir campos de dados privados.

A classe bem encapsulada deve ocultar seus dados e os detalhes de implementação do mundo exterior. Isso é denominado programação caixa preta. Usando o encapsulamento, a implementação do método pode ser alterada pelo autor da classe sem quebrar qualquer código existente fazendo uso dela. (fonte: www.macoratti.net)”

“As propriedades combinam aspectos de métodos e campos. Para o usuário de um objeto, uma propriedade parece ser um campo. Acessar a propriedade requer a mesma sintaxe. Para o implementador de uma classe, uma propriedade consiste em um ou dois blocos de código, que representam um acessador [get](#) e/ou um acessador [set](#). O bloco de código para o acessador get é executado quando a propriedade é lida. O bloco de código para o acessador set é executado quando um novo valor é atribuído à propriedade. Uma propriedade sem um acessador set é considerada como somente leitura. Uma propriedade sem um acessador get é considerada como somente gravação. Uma propriedade que tem os dois acessadores é leitura/gravação.”

(Fonte: <https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/using-properties>).

Prof. Roberto de Castro

Atividade prática EXE032C: O próximo projeto teremos campos somente com o atributo “set” (com tratamento de dados), campos somente leitura (get) e método “void” (sem retorno de informação).

INTERFACE GRÁFICA

USP - UNIVERSIDADE SOL E PRAIA TERMINAL DE CONSULTA DO ALUNO

Primeira Nota: Segunda Nota:

Calcular Média

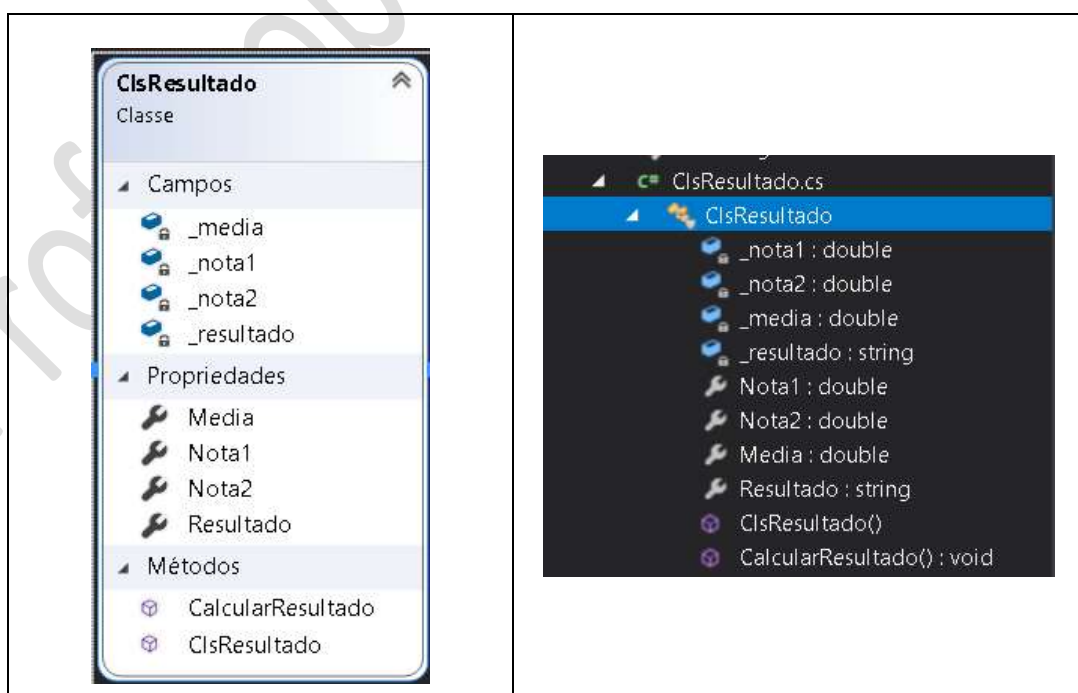
RESULTADO FINAL

Média Final:

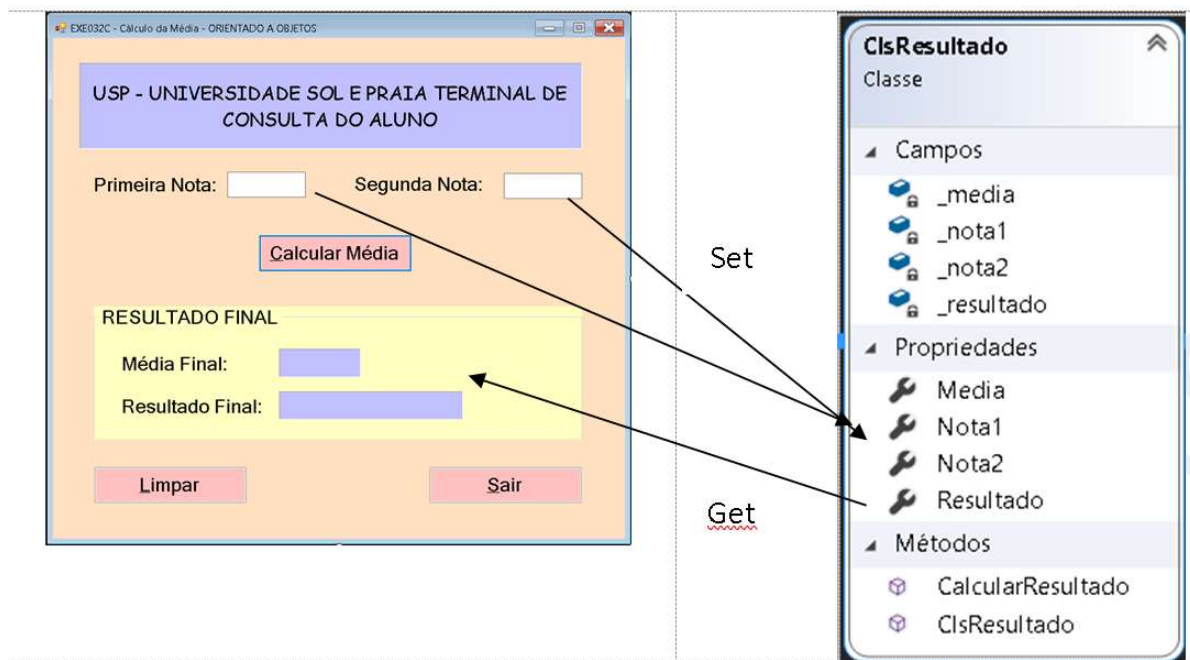
Resultado Final:

Limpar Sair

Diagrama de classe:



Prof. Roberto de Castro



Programação da classe:

```
using System;
```

```
namespace Exe011
```

```
{
```

```
    class ClsResultado
```

```
    {
```

```
        //Declaração dos campos/atributos
```

```
        private double _nota1;
```

```
        private double _nota2;
```

```
        private double _media;
```

```
        private string _resultado;
```

```
        //Declaração das propriedades gets/sets
```

```
        //Nota1 e Nota2 somente SET
```

```
        public double Nota1
```

```
        {
```

```
            set
```

```
            {
```

```
                if (value >= 0 && value <= 10)
```

```
                    _nota1 = value;
```

```
                else
```

```
                    throw new Exception("Atenção: Nota 1 fora do intervalo!!");
```

```
            }
```

```
        }
```

```
        public double Nota2
```

```
        {
```

```
set
{
    if (value >= 0 && value <= 10)
        _nota2 = value;
    else
        throw new Exception("Atenção: Nota 2 fora do intervalo!!");
}
```

//Propriedade Media e Resultado --> somente GET

```
public double Media { get => _media; }
public string Resultado { get => _resultado; }
```

//Método construtor

```
public ClsResultado()
{
    _nota1 = 0.0;
    _nota2 = 0.0;
    _media = 0.0;
    _resultado = "";
}
```

//Método calcular resultado (sem retorno)

```
public void CalcularResultado()
{
    _media = (_nota1 + _nota2) / 2;

    if (_media < 5)
    {
        _resultado = "Retido";
    }
    else
    {
        _resultado = "Promovido";
    }
}
```

Programação da interface gráfica (formulário):

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

namespace Exe011
{
    public partial class FrmExe011 : Form
    {
        //Declaração das variáveis
        double nota1 = 0;
```

```
double nota2 = 0;

public FrmExe011()
{
    InitializeComponent();
}

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

private void BtnLimpar_Click(object sender, EventArgs e)
{
    TxtNota1.Text = "";
    TxtNota2.Text = "";
    LblMedia.Text = "";
    LblResultado.Text = "";
    TxtNota1.Focus();
}

private void BtnCalcularMedia_Click(object sender, EventArgs e)
{
    try
    {
        nota1 = Convert.ToDouble(TxtNota1.Text);
        nota2 = Convert.ToDouble(TxtNota2.Text);

        //Cria a instância da classe
        ClsResultado ObjResultado = new ClsResultado();

        //Transfere as notas para a classe → SET
        ObjResultado.Nota1 = nota1;
        ObjResultado.Nota2 = nota2;

        //Observe que NÃO CONSEGUIMOS TRANSFERIR VALORES PARA OS CAMPOS GET
        //ObjResultado.Resultado = "Aprovado!!";

        //Executa o método calcular que não retorna nada!!
        ObjResultado.CalcularResultado();

        //Captura (recupera/pega) o conteúdo (GET) das propriedades Media e Resultado
        LblMedia.Text = ObjResultado.Media.ToString("0.0");
        LblResultado.Text = ObjResultado.Resultado;
    }
    catch (Exception ex)
    {
        MessageBox.Show("Erro --> " + ex.Message, "Atenção");
        TxtNota1.Focus();
    }
}
```

}

}

}

}

Prof. Roberto de Castro