

Escola Técnica Estadual “São Paulo” - ETESP

Linguagem C# - Visual Studio 2019

Overload

Overload significa sobrecarga de métodos. Métodos com os mesmos nomes, porém com “assinatura diferente”. **OBS:** Assinatura de um método é como ele foi declarado (se recebe parâmetros; se retorna com valores...), seu nome completo, nome e sobrenome (parâmetros).

Exercício proposto: EXE033 - Proposta de Orçamento

INTERFACE GRÁFICA

The screenshot shows a Windows application window titled "Exe033 - ORÇAMENTO - OOP". The interface has a purple header with the text "TACO LASCADO, SUJO E SOLTO" and "Manutenção e Limpeza de Pisos e Azulejos". Below this is an orange section titled "Orçamento" containing a text input for "Valor R\$:", a dropdown for "Num. Parcelas*" with the value "1", and a "Consultar" button. A note below reads "* 1 = A Vista com 15% desconto:". Below the orange section is a yellow section titled "Proposta de Pagamento" with a light blue text area. At the bottom are two yellow buttons: "Novo" and "Sair".

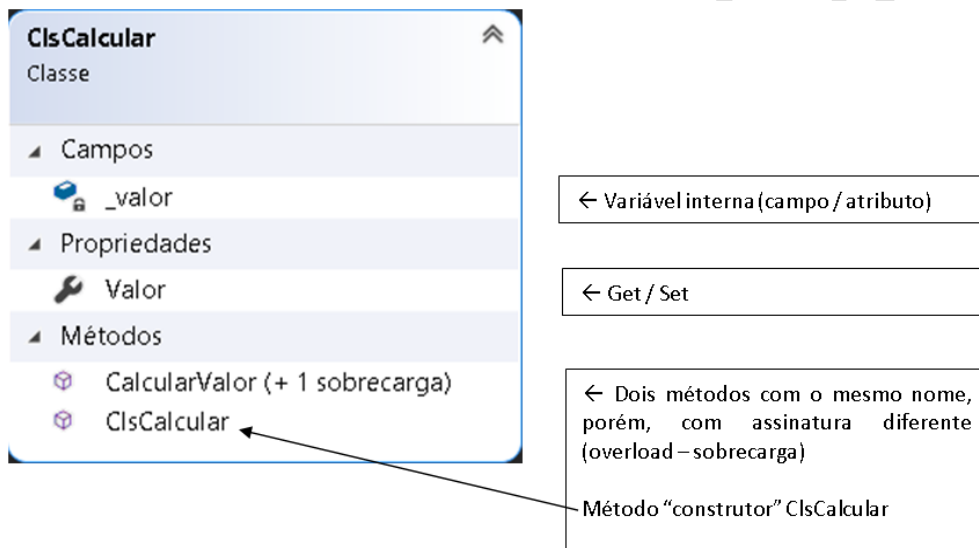
Execução do projeto:

Two side-by-side screenshots of the application. The left screenshot shows the "Orçamento" section with "Valor R\$:" set to "1000" and "Num. Parcelas*" set to "1". The "Proposta de Pagamento" section displays "O valor do orçamento com desconto a vista é de R\$850,00". The right screenshot shows the same interface but with "Num. Parcelas*" set to "3". The "Proposta de Pagamento" section displays "O valor será dividido em 3 parcelas de R\$333,33". Both screenshots have the same purple header and footer buttons ("Novo", "Sair").

Principais Etapas:

- Incluir a “classe” no projeto (Project → Add Class). Nome da Classe: ClsCalcular
- 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 da classe e o programa “cliente”. São elas que recebem e enviam dados.
- Declaração do construtor: O construtor é um método 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.

Veja o “diagrama da Classe”:



Programação da Classe “ClsCalcular”

```
namespace Exe033_OOP_Orcamento
{
    class ClsCalcular
    {
        //Declaração do atributo / campo
        decimal _valor = 0;

        //Declaração da propriedade
        public decimal Valor { get => _valor; set => _valor = value; }

        //Método construtor
        public ClsCalcular()
        {
            _valor = 0;
        }
    }
}
```

//Este é um exemplo para OverLoad (sobrecarga de métodos).
// Métodos com o mesmo nome, porém, com assinatura diferente!!!

//Método para parcela a vista

```
public string CalcularValor()
{
    decimal valorParcela = (_valor * 85) / 100;
    string texto = "O valor do orçamento com desconto a vista é de " + valorParcela.ToString("C2");
    //"C2" --> exibe o número no formato de dinheiro com duas casas decimais

    return texto;
}
```

//Método para pagamento parcelado. Mesmo nome, porém, com "assinatura diferente" (recebe parâmetro)

```
public string CalcularValor(int numParcela)
{
    decimal valorParcela = _valor / numParcela;

    string texto = "O valor será dividido em " + numParcela + " parcelas de " +
        valorParcela.ToString("C2");
    //"C2" --> exibe o número no formato de dinheiro com duas casas decimais

    return texto;
}
}
```

Programação do "projeto principal (formulário)"

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

namespace Exe033_OOP_Orcamento
{
    public partial class FrmExe033 : Form
    {
        public FrmExe033()
        {
            InitializeComponent();
        }

        private void FrmExe033_Load(object sender, EventArgs e)
        {
            //Preenche o ComboBox com as formas de pagamento
            for (int x = 1; x <= 6; x++)
            {
                CboFormaPagto.Items.Add(x); ;
            }
        }
    }
}
```

```
CboFormaPagto.SelectedIndex = 0;
}

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

private void BtnNovo_Click(object sender, EventArgs e)
{
    LblResultado.Text = "";
    TxtValor.Text = "";
    CboFormaPagto.SelectedIndex = 0;
    TxtValor.Focus();
}

private void BtnConsultar_Click(object sender, EventArgs e)
{
    //Cria a instância da Classe. Neste momento o método construtor é executado
    ClsCalcular ObjCalcular = new ClsCalcular();

    //Transfere o "valor" para a classe
    ObjCalcular.Valor = Convert.ToDecimal(TxtValor.Text);

    if (CboFormaPagto.SelectedIndex == 0)
    {
        //Executa o método CalcularValor, SEM PARAMETRO, para desconto a vista, que retorna um texto
        LblResultado.Text = ObjCalcular.CalcularValor();
    }
    else
    {
        //Executa o método CalcularValor, COM PARAMETRO, para pagamento parcelado, que retorna
        um texto
        LblResultado.Text = ObjCalcular.CalcularValor(CboFormaPagto.SelectedIndex+1);
    }
}
}
```

Perceba o aviso de sobrecarga que é exibido, ao iniciar a digitação do nome do método:

```
41 TxtValor.Focus();
42 }
43
44
45 private void BtnConsultar_Click(object sender, EventArgs e)
46 {
47     //Cria a instância da Classe. Neste momento o método construtor é executado
48     ClsCalcular ObjCalcular = new ClsCalcular();
49
50     //Transfere o "valor" para a classe
51     ObjCalcular.Valor = Convert.ToDecimal(TxtValor.Text);
52
53     if (CboFormaPagto.SelectedIndex == 0)
54     {
55         //Executa o método CalcularValor, SEM PARAMETRO, para desconto a vista, que retorna um texto
56         LblResultado.Text = ObjCalcular.CalcularValor();
57     }
58     else
59     {
60         //Executa o método CalcularValor, COM PARAMETRO, para pagamento em parcelas, que retorna um texto
61         LblResultado.Text = ObjCalcular.
62     }
63 }
64
65
66
```

CalcularValor string ClsCalcular.CalcularValor() (+ 1 sobrecarga)

Equals
GetHashCode
GetType
NumParcela
ToString
Valor

6.4 - Herança e Overrind (sobrescrita):

Herança → Pode ser definida como a capacidade de uma classe “herdar” atributos e comportamentos de uma outra classe. É utilizado quando precisamos de uma “NOVA” classe, que tenha todas as características de outra classe, com algumas modificações em seu comportamento, ou mesmo algumas funcionalidades adicionais.

Overrind (sobrescrita): Substituir o método existente da classe pai (classe base), por um novo método escrito na classe filha.

IMPORTANTE: Para ocorrer a sobrescrita, precisamos aplicar o conceito de Herança!!!

***** Você só pode usar o override em métodos que permitam serem sobrescritos!!! *****

A proposta agora é “aproveitarmos” a classe já existente (ClsOperacao) para o próximo exercício. Vamos fazer uma cópia do projeto anterior “Somar dois valores” para não perdermos o que já está pronto.

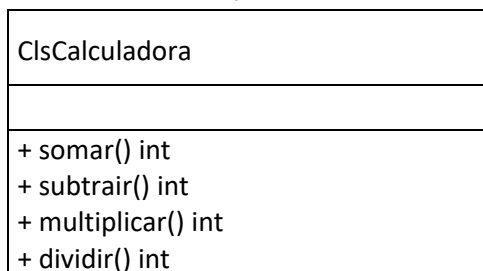
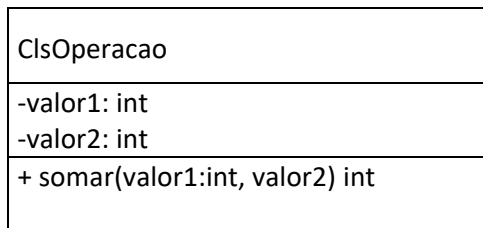
Nosso projeto já possui a classe “ClsOperacao” que faz a soma entre dois valores.

Neste novo projeto incluiremos uma nova classe: ClsOperacoes, que:

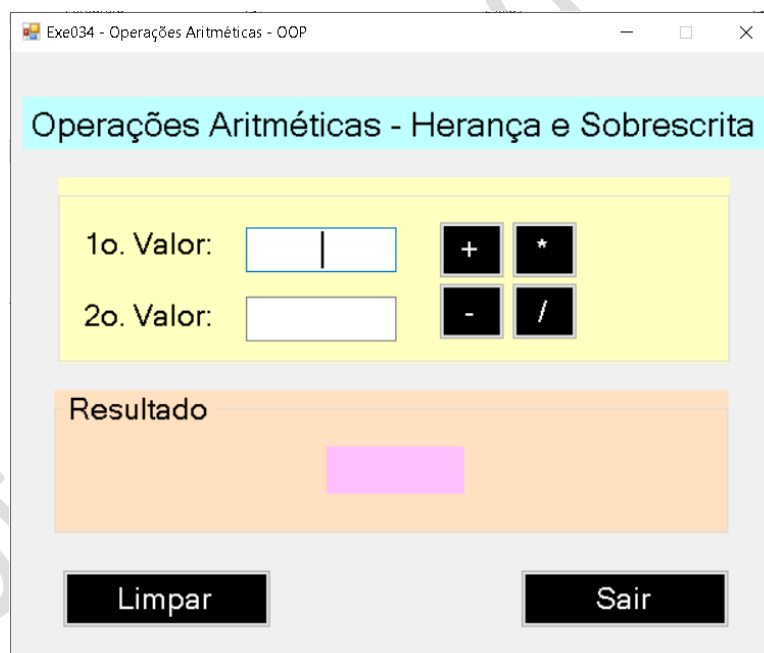
- 1) Herdará todos os atributos(campos), propriedades e métodos da classe “ClsOperacao”.
- 2) Irá “reescrever” o método “Somar” (exibirá uma mensagem, informando que está executando este novo método).
- 3) Terá “novos” métodos para efetuar a Subtração, Multiplicação e Divisão.

Prof. Roberto de Castro

Notação UML das classes:



Interface gráfica:



Programação da classe ClsOperacao (com MODIFICAÇÕES):

```
class ClsOperacao
{
    //Declaração dos atributos (variáveis) e dos gets e sets
    private int _valor1;
    private int _valor2;

    //Declaração das propriedades Gets / Sets
    public int Valor1 { get => _valor1; set => _valor1 = value; }
```

Prof. Roberto de Castro

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

```
//Declaração do Método Construtor - este método possui o mesmo nome da classe e NÃO possui VOID
```

```
//Irá inicializar os atributos - variáveis privativas
```

```
public ClsOperacao()  
{  
    _valor1 = 0;  
    _valor2 = 0;  
}
```

```
//Declaração do método que irá efetuar a soma entre os dois valores e devolver o resultado
```

```
//O termo "virtual" é necessário para que ocorra a sobrescrita. Este método será reescrito em
```

```
//nova classe
```

```
public virtual int Somar()  
{  
    return _valor1 + _valor2;  
}  
}  
}
```

Programação da Classe "ClsCalculos" (Que herda as propriedades e métodos da classe ClsOperacao)

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

```
namespace Exe032_OOP_Somar2Valores  
{
```

```
    //Observe que a classe ClsCalculos está "herdando" todos os métodos e as propriedades da classe ClsOperacao
```

```
    class ClsCalculos: ClsOperacao  
    {
```

```
        //Construtor
```

```
        public ClsCalculos()  
        {  
            //Será executado o construtor da classe base  
        }  
    }
```

```
    //Este método está fazendo a sobrescrita do método Somar existente na classe ClsOperacao
```

```
    public override int Somar()  
    {  
        MessageBox.Show ("É o método somar da classe ClsCalculos que está em execução!!");  
        return Valor1 + Valor2;  
    }  
}
```

```
    //Método para calcular a subtração e retornar o resultado
```

```
    public int Subtrair()
```

```
{  
    return Valor1 - Valor2;  
}  
  
//Método para calcular a multiplicação e retornar o resultado  
public int Multiplicar()  
{  
    return Valor1 * Valor2;  
}  
  
//Método para calcular a divisão e retornar o resultado  
public int Dividir()  
{  
    return Valor1 / Valor2;  
}  
}
```

Programação do Formulário:

```
using System;  
using System.Windows.Forms;  
  
namespace Exe032_OOP_Somar2Valores  
{  
    public partial class FrmExe034 : Form  
    {  
        public FrmExe034()  
        {  
            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();  
        }  
  
        //Este procedimento é executado pelos botões Soma, Subtração, Multiplicação e Divisão  
        private void BtnCalcular_Click(object sender, EventArgs e)  
        {  
            //Gera a instância da classe  
            ClsCalculos ObjOperacao = new ClsCalculos();  
        }  
    }  
}
```


Prof. Roberto de Castro

```
ObjOperacao.Valor1 = Convert.ToInt32(TxtValor1.Text);
ObjOperacao.Valor2 = Convert.ToInt32(TxtValor2.Text);
```

```
int resultado = 0;
```

```
Button botaoOperacao = sender as Button;
```

```
if (botaoOperacao.Text == "+")
{
    //Executa o "novo" método Somar da classe ClsCalculos
    resultado = ObjOperacao.Somar();
}
else if (botaoOperacao.Text == "-")
{
    //Executa o método Subtrair
    resultado = ObjOperacao.Subtrair();
}
else if (botaoOperacao.Text == "*")
{
    //Executa o método Multiplicar
    resultado = ObjOperacao.Multiplicar();
}
else
{
    //Executa o método Dividir
    resultado = ObjOperacao.Dividir();
}

//Exibe o resultado da operação
LblResultado.Text = resultado.ToString();
```

```
}
```

```
}
```

uma classe "incompleta"

Classe "Abstrata"

É uma classe "especial" que NÃO PODERÁ SER INSTANCIADA (poderá ser herdada), destinada a ser uma classe base para outras classes.

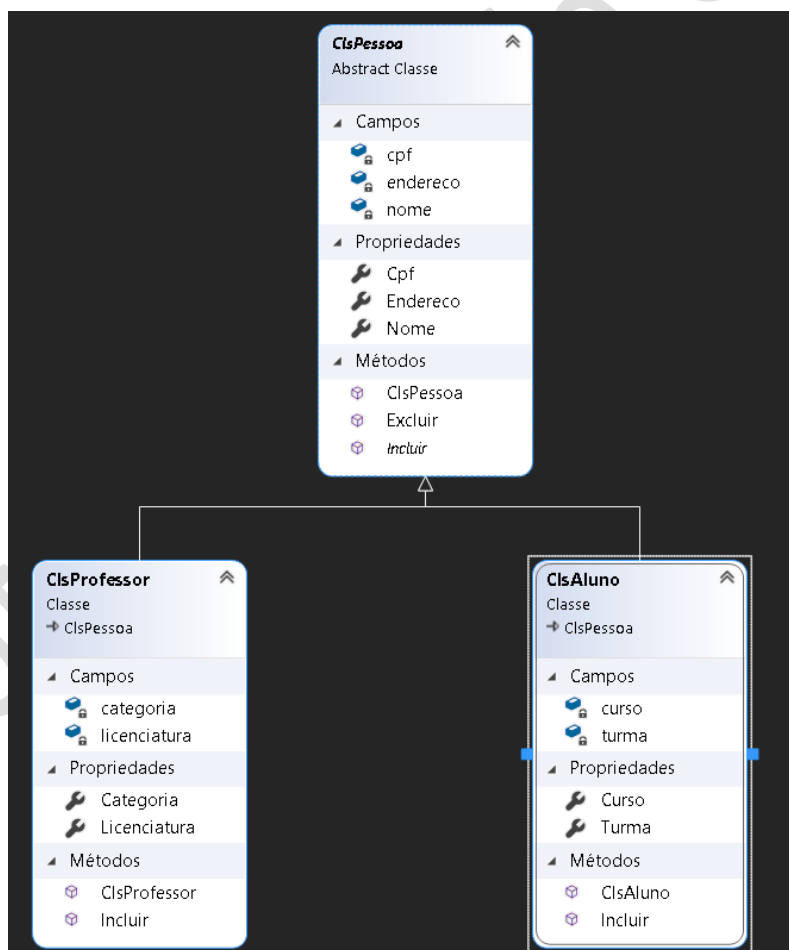
A classe abstrata poderá conter atributos, propriedades (gets/sets), métodos abstratos, métodos concretos ("normais") e o método construtor.

O método abstrato somente poderá ser declarado em uma classe abstrata. Quando declarado, o método "abstrato" não poderá ser implementado (conter "instruções"). DEVERÁ conter apenas a sua "assinatura", exemplo: `public abstract void Calcular();`

Sua implementação será obrigatória e deverá ser realizada na classe derivada, utilizando o modificador "override". Exemplo: `public override void Calcular();`

Vamos a um exemplo prático...

Observe o diagrama de classe abaixo:



o método Incluir na Classe Pessoa vai ser reescrito = o "incluir professor" vai ser diferente do "incluir aluno" (professor tem mais dois campos para preencher)

a partir da classe pessoas, vai implementar mais duas classes = professor e aluno
Ou seja, o professor e aluno são pessoas e tem CPF, endereço e nome, mas tem coisas distintas em cada classe (professor tem categoria e aluno tem curso, por exemplo)

classe abstrata = classe base pra outras subclasses - não consegue fazer instância (ClsPessoa); é um ponto de partida pra outras classes

classe aluno e professor herdando da classe pessoa

O projeto terá "dois formulários" (FrmMenu e FrmCadastro) e "três classes" (ClsPessoa, ClsProfessor e ClsAluno).

A classe "ClsPessoa" será "abstrata", terá o método Incluir (abstrato) e o método Excluir (concreto).

Prof. Roberto de Castro

As classes `ClsProfessor` e `ClsAluno` serão classes concretas, que herdarão da classe base `ClsPessoa`, os atributos, propriedades e métodos e que, OBRIGATORIAMENTE, deverão implementar o método abstrato "Incluir".

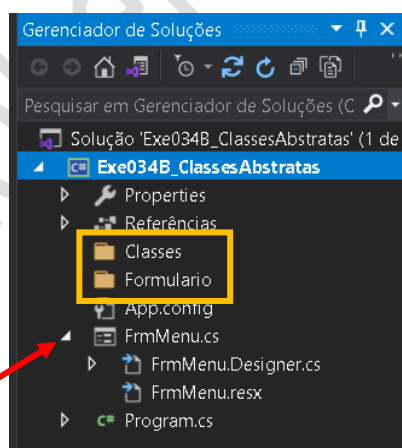
No Visual Studio, crie um novo projeto (`Exe034_ClassesAbstratas`) e altere as seguintes propriedades do formulário:

CONTROLE	PROPRIEDADE	CONTEÚDO
Form	Name	FrmMenu
	MaximizeBox	False
	StartPosition	CenterScreen
	Text	Classes Abstradas
	WindowState	Maximized
	IsMdiContainer	True

Antes de prosseguirmos com a configuração deste formulário, vamos "organizar" nosso projeto com a criação de duas pastas: uma que irá conter os formulários e outra que irá conter as classes.

Na janela do "Gerenciador de Soluções", pressione o botão direito do mouse sobre o nome do projeto e selecione a opção: Adicionar → Nova Pasta. Dê o nome de "Formulario" (sem acento). Repita a operação mais uma vez: Pressione novamente o botão direito do mouse sobre o nome projeto e selecione a opção: Adicionar → Nova Pasta. Dê o nome de "Classes".

Após a criação das pastas, o "Solution Explorer" deverá estar com a aparência abaixo:

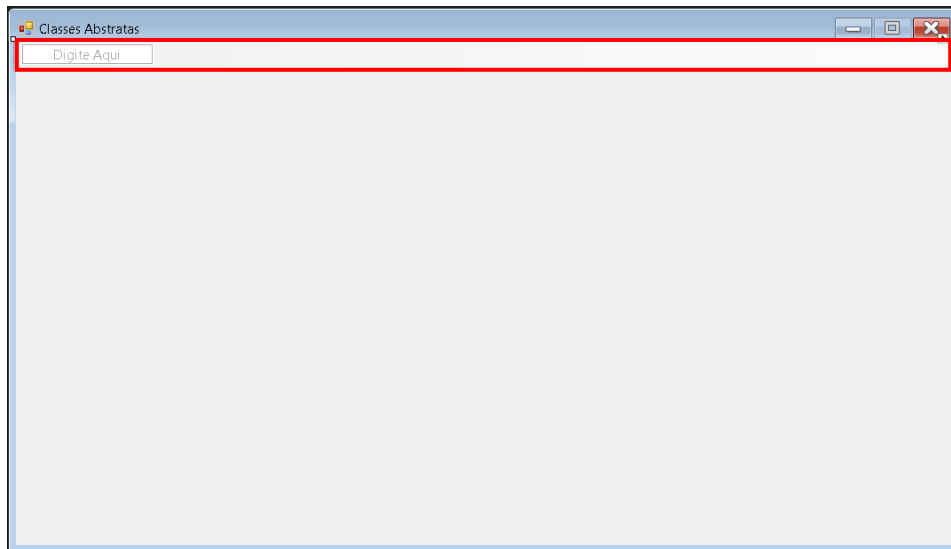


Clique no "FrmMenu" e "arraste" para a pasta "Formulario" (será exibida uma mensagem solicitando a confirmação). O formulário "FrmMenu" será transferido para a pasta "Formulario".

Faça um "pequeno" teste, execute o projeto para confirmar se o funcionamento está ok! O formulário DEVERÁ ser exibido maximizado.

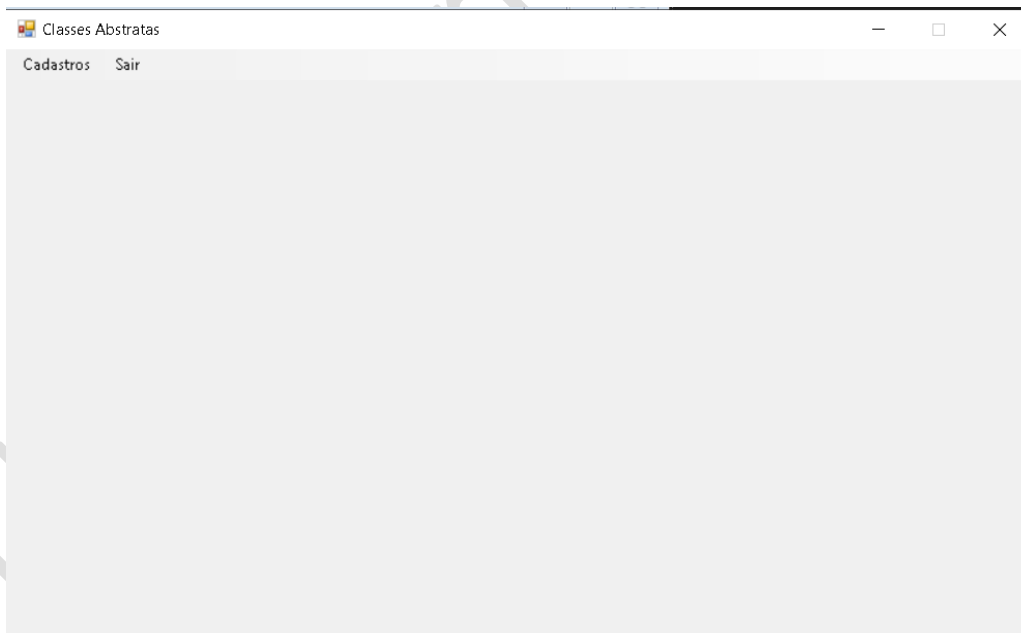
Se tudo ok, prosseguiremos com a configuração do formulário "Menu", que está dentro da pasta "Formulario".

Vamos inserir neste formulário o controle "MenuStrip". Este controle será inserido automaticamente no "topo" do formulário.



Vamos incluir neste menu, duas opções: Cadastros e Sair.

O menu deverá ficar como o modelo abaixo:



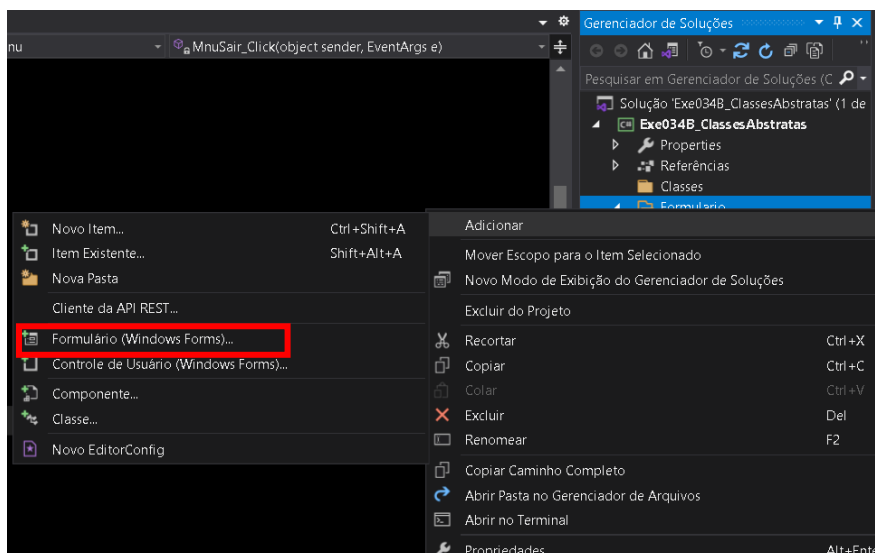
Altere a propriedade “name” destes “menus” para: MnuCadastros e MnuSair.

Inclua a programação no menu “Sair”. Dê dois cliques sobre o texto “Sair” e insira a programação:

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

}

Vamos incluir o segundo formulário... Botão direito sobre a pasta “Formulario” e selecione a opção: Adicionar
→ Formulario



Para este segundo formulário dê o nome de FrmCadastro.

Altere as propriedades abaixo:

CONTROLE	PROPRIEDADE	CONTEÚDO
Form	Name	FrmCadastro
	MaximizeBox	False
	StartPosition	CenterScreen
	Text	Cadastros

Inclua o controle “TabControl” (funcionará como “abas”).

Vamos configurar as “abas”. Selecione o controle “TabControl” e clique na propriedade “TabPage” e altere, respectivamente, o “text” para “Professores” e “Alunos”.

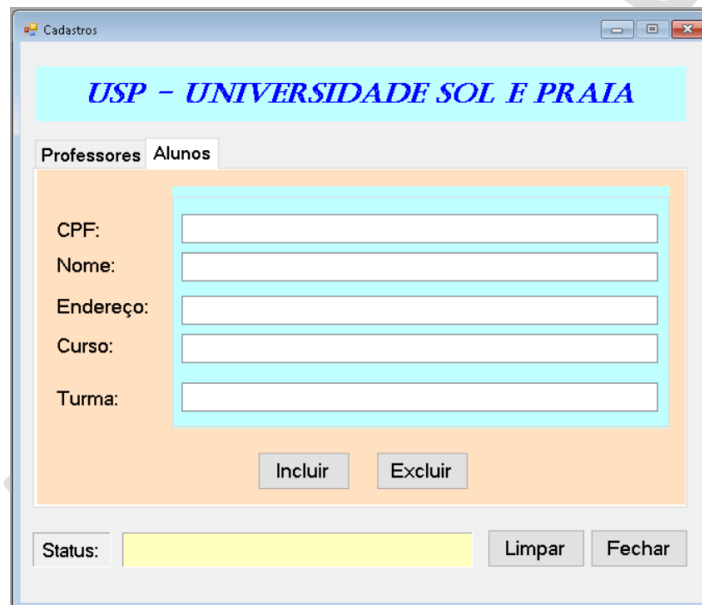
Na aba “Professores”, inclua os controles abaixo:

Prof. Roberto de Castro



The screenshot shows a Windows-style application window titled "Cadastro". At the top, there is a blue header bar with the text "USP - UNIVERSIDADE SOL E PRAIA" in blue. Below the header, there are two tabs: "Professores" (selected) and "Alunos". The main area has a light green background. It contains five text input fields labeled "CPF:", "Nome:", "Endereço:", "Licenciatura:", and "Categoria:". Below these fields are two buttons: "Incluir" and "Excluir". At the bottom of the window, there is a "Status:" label followed by a yellow rectangular field, and two more buttons: "Limpar" and "Fechar".

Na aba "Alunos":



The screenshot shows the same "Cadastro" application window, but with the "Alunos" tab selected. The main area now has a light orange background. The text input fields are labeled "CPF:", "Nome:", "Endereço:", "Curso:", and "Turma:". The "Incluir" and "Excluir" buttons are still present. The "Status:" field and "Limpar" and "Fechar" buttons remain at the bottom.

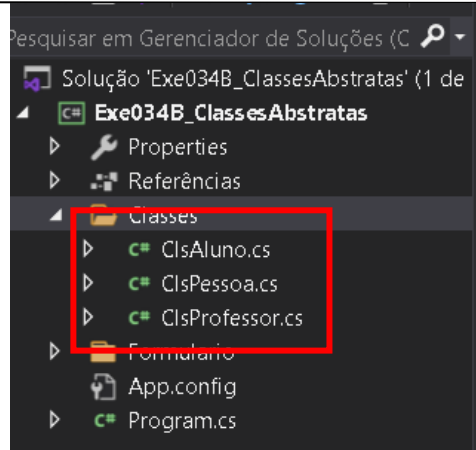
Finalizada a etapa de criação e configuração dos formulários, vamos agora para a programação das classes!!

Vamos incluir três classes. Clique com o botão direito do mouse sobre a pasta "Classes", selecione a opção Adicionar → Classe.

A primeira classe se chamará "ClsPessoa", a segunda "ClsProfessor" e a terceira "ClsAluno".

Após a criação das três classes, a pasta "Classes" deverá estar conforme modelo abaixo:

Prof. Roberto de Castro



Vamos incluir a programação da classe “ClsPessoa” (classe abstrata que será a classe base para as demais).

```
namespace Exe034B_ClassesAbstratas.Classes
{
    public abstract class ClsPessoa
    {
        //Declaração dos atributos - campos
        private string cpf;
        private string nome;
        private string endereco;

        //Declaração das propriedades (gets / sets)
        public string Cpf { get => cpf; set => cpf = value; }
        public string Nome { get => nome; set => nome = value; }
        public string Endereco { get => endereco; set => endereco = value; }

        //Método construtor
        public ClsPessoa()
        {
            cpf = "";
            nome = "";
            endereco = "";
        }

        //Método abstrato que DEVERÁ ser implementado nas subclasses: Aluno e Professor
        public abstract string Incluir();
    }
}
```

```
//Método CONCRETO que será herdado pelas subclasses: Aluno e Professor
public string Excluir()
{
    return "Dados excluídos com sucesso!!!";
}
}
```

Classe "ClsProfessor" (que herda a classe abstrata "ClsPessoa"). Observe o destaque em "vermelho":

```
namespace Exe034B_ClassesAbstratas.Classes
{
    public class ClsProfessor : ClsPessoa
    {
        //Declaração dos atributos / campos
        private string licenciatura;
        private string categoria;

        //Declaração das propriedades
        public string Licenciatura { get => licenciatura; set => licenciatura = value; }
        public string Categoria { get => categoria; set => categoria = value; }

        //Declaração do construtor
        public ClsProfessor()
        {
            licenciatura = "";
            categoria = "";
        }

        //implementação do método abstrato que está classe base - ClsPessoa
        public override string Incluir()
        {
            return "Dados do professor incluídos com sucesso:\n"
                + Cpf + "\n"
                + Nome + "\n"
                + Endereco + "\n"
                + Licenciatura + "\n"
                + Categoria;
        }
    }
}
```

Classe "ClsAluno" (que herda a classe abstrata "ClsPessoa"). Observe o destaque em "vermelho":

```
namespace Exe034B_ClassesAbstratas.Classes
{
    //Observe os dois pontos ":" após o nome da classe ClsAluno
    //--> indica que a classe ClsAluno está herdando da classe ClsPessoa
    public class ClsAluno : ClsPessoa
    {
        //...
    }
}
```



```
{
    //Declaração dos atributos / campos
    private string curso;
    private string turma;

    //Declaração das propriedades
    public string Curso { get => curso; set => curso = value; }
    public string Turma { get => turma; set => turma = value; }

    //Declaração do construtor
    public ClsAluno()
    {
        curso = "";
        turma = "";
    }

    //implementação do método abstrato que está classe base - ClsPessoa
    public override string Incluir()
    {
        return "Dados do aluno incluídos com sucesso:\n"
            + Cpf + "\n"
            + Nome + "\n"
            + Endereco + "\n"
            + Curso + "\n"
            + Turma;
    }
}
```

Programação do formulário:

```
using System;
using System.Windows.Forms;
using Exe034B_ClassesAbstratas.Classes; //using necessário para acessar as Classes

namespace Exe034B_ClassesAbstratas.Formulario
{
    public partial class FrmCadastro : Form
    {
        public FrmCadastro()
        {
            InitializeComponent();
        }

        private void BtnFechar_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void BtnIncluir_Click(object sender, EventArgs e)
        {

```

Prof. Roberto de Castro

```
//Cria a instância da classe Professor
//Observe a inclusão do using nas linhas iniciais...
//
ClsProfessor ObjProfessor = new ClsProfessor();

//Envia informações para a classe professor
ObjProfessor.Cpf = TxtCpfProfessor.Text;
ObjProfessor.Nome = TxtNomePro.Text;
ObjProfessor.Endereco = TxtEnderecoPro.Text;
ObjProfessor.Licenciatura = TxtLicenciatura.Text;
ObjProfessor.Categoria = TxtCategoria.Text;

//executa o método incluir (sobrescrito) da classe Professor
LblStatus.Text = ObjProfessor.Incluir();
}

private void BtnExcluir_Click(object sender, EventArgs e)
{
    //Cria a instância da classe Professor
    ClsProfessor ObjProfessor = new ClsProfessor();

    //Executa o método Excluir da classe base
    if (TxtCpfProfessor.Text != "")
    {
        //Envia informações para a classe professor
        ObjProfessor.Cpf = TxtCpfProfessor.Text;
        LblStatus.Text = ObjProfessor.Excluir();
    }
    else
    {
        LblStatus.Text = "ATENÇÃO: Digite o número do CPF para exclusão!!";
    }
}

private void BtnIncluirAluno_Click(object sender, EventArgs e)
{
    //Cria a instância da classe Aluno
    //Importante: Observe a inclusão do "using" nas linhas iniciais
    //
    ClsAluno ObjAluno = new ClsAluno();

    //Envia informações para a classe Aluno
    ObjAluno.Cpf = TxtCpfAluno.Text;
    ObjAluno.Nome = TxtNomeAluno.Text;
    ObjAluno.Endereco = TxtEnderecoAluno.Text;
    ObjAlunoCurso = TxtCurso.Text;
    ObjAluno.Turma = TxtTurma.Text;

    //executa o método incluir (sobrescrito) da classe Aluno
    LblStatus.Text = ObjAluno.Incluir();
}
```

```
private void BtnExcluirAluno_Click(object sender, EventArgs e)
{
    //Cria a instância da classe Aluno
    ClsAluno ObjAluno = new ClsAluno();

    //Executa o método Excluir da classe base
    if (TxtCpfAluno.Text != "")
    {
        //Envia informações para a classe Aluno
        ObjAluno.Cpf = TxtCpfAluno.Text;
        LblStatus.Text = ObjAluno.Excluir();
    }
    else
    {
        LblStatus.Text = "ATENÇÃO: Digite o número do CPF para exclusão!!";
    }
}

private void BtnLimpar_Click(object sender, EventArgs e)
{
    if (tabControl1.SelectedIndex == 0) //a guia Professores está ativa
    {
        foreach (TextBox CaixaTexto in GrpProfessor.Controls)
        {
            CaixaTexto.Text = "";
        }
    }
    else //a guia Alunos está ativa
    {
        foreach (TextBox caixaTexto in GrpAluno.Controls)
        {
            caixaTexto.Text = "";
        }
    }

    LblStatus.Text = "";
}
}
```