

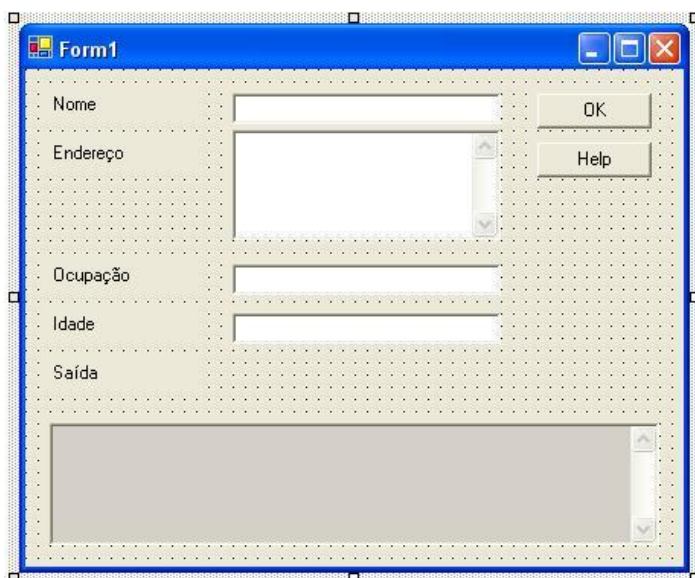
Aplicações em Windows Forms usando a Linguagem C#

Projeto TextBoxTeste

Criaremos uma caixa de diálogo na qual você poderá digitar seu nome, endereço, profissão/ocupação e idade. O objetivo deste exemplo é dar a você uma boa fundamentação sobre a manipulação de propriedades e o uso de eventos para depois criarmos algo extremamente útil, isto é, quando dominemos estes eventos, poderemos aprofundar nossos conhecimentos em Windows Forms.

Em primeiro lugar, construímos a interface do usuário:

1. Selecione *File / New* e crie um novo Aplicativo Windows (Windows Application) em Visual C# Projects. Chame o projeto de **TextBoxTeste**.
2. Crie o formulário que é mostrado abaixo, arrastando os labels, caixas de texto e botões para a superfície do projeto. Antes de poder redimensionar as duas caixas de texto, defina a propriedade **Multiline** para **true**. Faça isso, já seja na janela de Propriedades ou dando um clique direito nos controles e selecionando **Properties**.



3. Dê os nomes aos controles, como mostra a tabela abaixo:

Controle	Nome do Controle	Texto do Controle
label1	lblNome	Nome
label2	lblEndereco	Endereço
label3	lblOcupacao	Ocupação
label4	lblIdade	Idade
label5	lblSaida	Saída
textBox1	txtNome	

textBox2	txtEndereco	
textBox3	txtOcupacao	
textBox4	txtIdade	
textBox5	txtSaida	
button1	btnOK	OK
button2	btnHelp	Help


4. Defina a propriedade **Scrollbars** para os dois controles **txtSaida** e **txtEndereco** para **Vertical**.
5. Defina a propriedade **ReadOnly** do controle **txtSaida** para **true**.
6. Defina a propriedade **CausesValidation** do botão **btnHelp** para **false**. Lembrar que vimos e falamos sobre os eventos **Validating** e **Validate**. Se os definirmos para **false**, permitirá que o usuário dê um clique neste botão sem precisar ser incomodado sobre dados inválidos.
7. Quando você tiver dimensionado o formulário para encaixar-se perfeitamente nos controles, é hora de “ancorar” os controles de forma que eles se comportem adequadamente quando o formulário for redimensionado. Defina a propriedade **Anchor** como é mostrado na figura abaixo:

Controle	Valor de âncora
Todos os labels (etiquetas)	Top, Bottom, Left
Todas as caixas de texto (textbox) exceto txtEndereco	Top, Bottom, Left, Right
A caixa de texto de endereço (txtEndereco)	Top, Left, Right
Ambos os botões	Top, Right

8. Há uma última definição a ser feita. No formulário, encontrar as propriedades **Size** e **MinimumSize**. Preencha a propriedade **MinimumSize** para o tamanho do formulário. Além disso, queremos que o formulário não apareça em qualquer lugar da tela, que sempre apareça no centro da tela. Para isso, na seção **Layout**, em **StartPosition**, mudar a propriedade de **WindowsDefaultLocation** para **CenterScreen**, posicionando o nosso formulário sempre no centro da tela, toda vez que abrirmos e executarmos o formulário, evitando a disposição aleatória na tela e fazendo que o formulário fique melhor disposto.

O trabalho de definir a parte visual do formulário está finalizado agora. Ao executar o programa, nada acontecerá quando você der um clique nos botões ou digitar o texto, mas se maximizar ou puxar a caixa de diálogo, os controles se comportarão exatamente como você quer que eles façam em uma interface de usuário adequada, permanecendo fixos e redimensionando para preencher toda a caixa de diálogo. Se você já tentou realizar a mesma tarefa em uma linguagem como o Visual Basic 6 ou o Delphi até a versão 7, sabe quanto tempo economizou.

Podemos dar uma olhada no código. Você pode dar um clique direito no formulário e selecionar a opção **View Code** (Exibir código) ou usar a tecla de

atalho **F7**. Se você tiver fixado a caixa de ferramentas, você deverá remover a tachinha  para abrir mais espaço para a janela de código.

Surpreendentemente, é muito pouco o código visível no editor. No topo da nossa classe, os controles são definidos, mas o código não é visível até que você expanda a região rotulada **Windows Form Generated Code** e assim você poderá ver para onde foi todo seu trabalho.

Deve ser dito e enfatizado que você NUNCA deve editar o código nesta seção ! Da próxima vez que alterar alguma coisa no modo *designer*, ele será sobrescrito ou, o que é pior ainda, você poderá alterar alguma coisa de modo que o Form Designer não possa mais exibir o formulário.

Sugiro que vocês leiam o material da MSDN Library (disponível em seus computadores de trabalho) e/ou artigos da *MSDN Brasil* (<http://www.msdnbrasil.com.br>) , *Linha de Código* (<http://www.linhadecodigo.com.br>), *Codificando* (<http://www.codificando.net>), C-Sharp BR (<http://www.chsharpbr.com.br>) ou finalmente os materiais em inglês disponíveis em www.gotdotnet.com ou em www.windowsforms.com ou em www.4guysfromrolla.com para aprofundar seus conhecimentos e poder ter uma maior e melhor noção para seus conhecimentos se aprimorarem neste setor em especial do .NET com o C#.

Acrescentando os manipuladores de eventos

Volte para o **Form Designer** dando um clique no topo do editor de texto ou simplesmente usando o atalho **Shift + F7** e, no formulário, dê um clique duplo no botão **btnOK**. Da mesma forma faremos para o botão **btnHelp**. O código para o evento do botão OK é:

```
private void btnOK_Click(object sender, System.EventArgs e)
{
    //Nenhum teste de valor inválido é realizado
    //pois não deve ser necessário

    string output;

    //Concatena os valores de texto das quatro cixas de texto
    output = "Nome: " + this.txtNome.Text + "\r\n";
    output += "Endereço: " + this.txtEndereco.Text + "\r\n";
    output += "Ocupação: " + this.txtOcupacao.Text + "\r\n";
    output += "Idade: " + this.txtIdade.Text;

    //Insere o novo texto
    this.txtSaida.Text = output;
}
```

Para o evento do botão Help, segue o código abaixo:

```
private void btnHelp_Click(object sender, System.EventArgs e)
{
    //Escreve uma breve descrição de cada caixa de texto na caixa de texto de saída
    string output;

    output = "Nome = Seu Nome" + "\r\n";
    output += "Endereço = Seu endereço " + "\r\n";
    output += "Ocupação = único valor permitido é 'Programador' ou vazio " + "\r\n";
    output += "Idade = Sua Idade " ;

    //Insere o novo texto
    this.txtSaida.Text = output;
}
```

Em ambas as funções, as propriedades **Text** das caixas de texto são usadas, quer sejam recuperadas ou definidas na função **btnOK_Click()** ou simplesmente definidas como na função **btnHelp_Click**.

Nós inserimos as informações que o usuário digitou sem nos preocuparmos se elas estavam corretas. Isso significa que devemos realizar a verificação em outro lugar. Neste exemplo, existem vários critérios que devem ser cumpridos para que o valor seja correto:

- ▶ O nome do usuário não pode estar vazio.
- ▶ A idade do usuário deve ser um numero maior ou igual a zero.
- ▶ A ocupação do usuário deverá ser 'Programador' ou permanecer vazia.
- ▶ O endereço do usuário não pode ficar vazio.

A partir disto, podemos notar que a verificação que deve ser feita em duas caixas de texto (**txtNome** e **txtEndereco**) é a mesma. Nós também devemos evitar que o usuário digite qualquer coisa inválida na caixa **Idade** e, finalmente, devemos verificar se ele é um programador.

Para evitar que o usuário dê um clique em **OK** antes que qualquer coisa seja digitada, começamos definindo a propriedade **Enabled** do botão **OK** para **false** no construtor do nosso formulário, certificando-nos de não definir a propriedade até depois que o código criado em **InitializeComponent()** tenha sido chamado.

```
public Form1()
{
    //
    // Required for Windows Form Designer support
    //
    InitializeComponent();
    this.btnOK.Enabled = false;
}
```

Agora, nós criaremos o manipulador para as duas caixas de texto que devem ser verificadas para ver se não estão vazias. Fazemos isto realizando a assinatura do evento **Validating** das caixas de texto. Nós informamos o controle de que o evento deve ser manipulado por um método chamado **txtBoxVazio_Valida()**.

Precisamos também, de uma forma de saber o estado de nossos controles. Com esse objetivo usaremos a propriedade **Tag** das caixas de texto. Se lembrarmos o que foi falado sobre **Tag** anteriormente, apenas strings podem ser atribuídas à propriedade **Tag** a partir do Forms Designer. Contudo, como estamos definindo o valor **Tag** a partir do código, podemos fazer quase tudo o que quisermos com ele, e é mais adequado entrar um valor lógico (booleano) aqui.

No construtor, acrescentamos as seguintes instruções:

```
this.btnOK.Enabled = false;
//Valores da tag para testar se os dados são válidos
this.txtEndereco.Tag = false;
this.txtIdade.Tag = false;
this.txtNome.Tag = false;
this.txtOcupacao.Tag = false;
//Assinatura aos eventos
this.txtNome.Validating += new System.ComponentModel.CancelEventHandler(this.txtBoxVazio_Valida);
this.txtEndereco.Validating += new System.ComponentModel.CancelEventHandler(this.txtBoxVazio_Valida);
```

Ao contrário do manipulador de eventos do botão que vimos anteriormente, o manipulador de eventos para o evento **Validating** é uma versão especializada do manipulador padrão **System.EventHandler**.

O motivo pelo qual esse evento precisa de um manipulador especial é que, caso a validação falhe, deve haver um meio de evitar qualquer processamento posterior. Se fôssemos cancelar o processamento seguinte, isto significaria efetivamente que seria impossível sair da caixa de texto até que os dados digitados fossem válidos. Não faremos nada tão radical quanto isso neste exemplo.

Acrescentamos o manipulador de eventos, como segue :

```
private void txtBoxVazio_Valida(object sender, System.ComponentModel.CancelEventArgs e)
{
    TextBox tb;

    tb = (TextBox)sender;
    //Se o texto está vazio, nós mudamos a cor de fundo da caixa
    //de texto para vermelho para indicar o problema. Usamos o
    //valor da tag do controle para indicar se o controle contém
    //uma informação válida
    if (tb.Text.Length == 0)
    {
        tb.BackColor = Color.Red;
        tb.Tag = false;
    }
    else
    {
        tb.BackColor = System.Drawing.SystemColors.Window;
        tb.Tag = true;
    }
    //Finalmente, chamamos ValidarTudo, que irá
    //configurar o valor do botão OK.
    ValidarTudo();
}
```

Como mais de uma caixa de texto está usando este método para manipular eventos, nós não podemos ter certeza sobre qual delas está chamando a

função. Sabemos, porém, que o efeito de chamar o método deve ser o mesmo, não importando quem o chama, portanto, podemos simplesmente converter (*fazer o cast*) o parâmetro emissor para uma caixa de texto e trabalhar com isto.

Se a extensão de texto na caixa de texto for zero (0), definimos a cor de fundo para vermelho e a **Tag** para **false**. Caso contrário, definimos a cor de fundo para a cor padrão do Windows para uma janela.

Obs: Você deve sempre usar as cores que podem ser encontradas na enumeração **System.Drawing.SystemColor** quando quiser definir uma cor padrão em um controle. Se você simplesmente definir a cor para branco, o seu aplicativo aparecerá de uma forma estranha se o usuário tiver alterado as definições – padrão de cores.

Prosseguindo com o evento **Validating**, o próximo manipulador que acrescentaremos é para a caixa de texto *Ocupação*. O procedimento é exatamente o mesmo que para os dois manipuladores anteriores, mas o código de validação é diferente, pois a ocupação deve ser *Programador* ou uma string vazia para ser válida. Portanto, acrescentaremos uma nova linha ao construtor.

```
this.txtEndereco.Validating += new System.ComponentModel.CancelEventHandler(this.txtBoxVazio_Valida);  
this.txtOcupacao.Validating += new System.ComponentModel.CancelEventHandler(this.txtBoxVazio_Valida);
```

ou, escrito em letras maiores:

```
this.txtOcupacao.Validating += new  
System.ComponentModel.CancelEventHandler(this.txtBoxVazio_Valida);
```

E depois, após o `private void txtBoxVazioValida ...` adicionar:

```
    {  
        tb.BackColor = System.Drawing.SystemColors.Window;  
        tb.Tag = true;  
    }  
    //Finalmente, chamamos ValidaTudo, que irá  
    //configurar o valor do botão OK.  
    ValidaTudo();  
}  
  
private void txtOcupacao_Valida(object sender, System.ComponentModel.CancelEventArgs e)  
{  
    //Faz o cast do objeto sender para a caixa de texto  
    TextBox tb = (TextBox)sender;  
  
    //Verifica se os valores são corretos  
    if (tb.Text.CompareTo("Programador") == 0 || tb.Text.Length == 0)  
    {  
        tb.Tag = true;  
        tb.BackColor = System.Drawing.SystemColors.Window;  
    }  
    else  
    {  
        tb.Tag = false;  
        tb.BackColor = Color.Red;  
    }  
  
    //Configura o estado do botão OK  
    ValidaTudo();  
}
```

O nosso segundo desafio é a caixa de texto *idade*. Desejamos que o usuário digite apenas números positivos (inclusive 0 para tornar o teste mais simples). Para obtermos isto, nós usaremos o evento **KeyPress** para remover quaisquer caracteres não desejados antes que eles sejam exibidos na caixa de texto.

Primeiro, fazemos a assinatura do evento **KeyPress**. É só proceder como nos manipuladores de evento anteriores no construtor:

```
//Assinatura aos eventos
this.txtNome.Validating += new System.ComponentModel.CancelEventHandler(this.txtBoxVazio_Valida);
this.txtEndereco.Validating += new System.ComponentModel.CancelEventHandler(this.txtBoxVazio_Valida);
this.txtOcupacao.Validating += new System.ComponentModel.CancelEventHandler(this.txtOcupacao_Valida);
this.txtIdade.KeyPress += new System.Windows.Forms.KeyPressEventHandler(this.txtIdade_PressionaTecla);
```

Em que é adicionada a linha, que, em letras maiores, fica assim:

```
this.txtIdade.KeyPress += new
System.Windows.Forms.KeyPressEventHandler(this.txtIdade_PressionaTecla
);
```

Esse manipulador de eventos também é especializado.

System.Windows.Forms.KeyPressEventHandler é o evento fornecido porque o evento precisa de informações sobre a tecla que foi pressionada.

Acrescentamos, então, o próprio manipulador de eventos:

```
private void txtIdade_PressionaTecla(object sender, System.Windows.Forms.KeyPressEventArgs e)
{
    if ((e.KeyChar < 48 || e.KeyChar > 57) && e.KeyChar != 8)
        e.Handled = true; //Remove o caractere
}
```

Os valores ASCII para os caracteres entre 0 e 9 estão entre 48 e 57, portanto certifique-se de que o caractere esteja dentro deste intervalo. Fazemos uma exceção, porém. O valor ASCII 8 é a tecla de **Backspace** (Voltar) e para facilitar a edição, nós permitimos que ela passe.

A definição da propriedade **Handled** de **KeyPressEventArgs** para **true** diz ao console que ele não deve fazer mais nada com o caractere e, portanto, ele não é mostrado.

Como está agora, o controle não está marcado como válido ou inválido. Isto acontece porque precisamos de outra verificação para nos certificarmos de que alguma coisa foi digitada.

Isto é algo simples, visto que já escrevemos o método para realizar esta verificação e nós simplesmente fizemos a assinatura ao evento **Validating** para o controle *Idade*, além de acrescentar a seguinte linha ao construtor:

```
//Assinatura aos eventos
this.txtNome.Validating += new System.ComponentModel.CancelEventHandler(this.txtBoxVazio_Valida);
this.txtEndereco.Validating += new System.ComponentModel.CancelEventHandler(this.txtBoxVazio_Valida);
this.txtOcupacao.Validating += new System.ComponentModel.CancelEventHandler(this.txtOcupacao_Valida);
this.txtIdade.KeyPress += new System.Windows.Forms.KeyPressEventHandler(this.txtIdade_PressionaTecla);
this.txtIdade.Validating += new System.ComponentModel.CancelEventHandler(this.txtBoxVazio_Valida);
```

Ou, em letras maiores:

```
this.txtIdade.Validating += new
System.ComponentModel.CancelEventHandler(this.txtBoxVazio_Valida);
```

Um ultimo caso deve ser manipulado para todos os controles. Se o usuário tiver digitado texto válido em todas as caixas de texto e, então, alterar algo, tornando o texto inválido, o botão OK permanece ativado. Assim, precisamos manipular um último evento para todas as caixas de texto: o evento **Change** que desligará o botão OK caso qualquer campo de texto contenha dados inválidos.

O evento **Change** é gerado sempre que o texto no controle é alterado. Fazemos a assinatura ao evento acrescentando as seguintes linhas ao construtor:

```
this.txtIdade.Validating += new System.ComponentModel.CancelEventHandler(this.txtBoxVazio_Valida);
this.txtNome.TextChanged += new System.EventHandler(this.txtBox_TextoMudado);
this.txtEndereco.TextChanged += new System.EventHandler(this.txtBox_TextoMudado);
this.txtIdade.TextChanged += new System.EventHandler(this.txtBox_TextoMudado);
this.txtOcupacao.TextChanged += new System.EventHandler(this.txtBox_TextoMudado);
```

Obs.: As linhas que devem ser adicionadas no código são as 4 últimas no código acima. Cuidado com adicionar a primeira linha, que já está no código

O evento **Change** usa o manipulador de eventos padrão que nós conhecemos no evento **Click**. Finalmente, acrescentamos o próprio evento:


```

        if ((e.KeyChar < 48 || e.KeyChar > 57) && e.KeyChar != 8)
        {
            e.Handled = true; //Remove o caractere
        }

        private void txtBox_TextoMudado(object sender, System.EventArgs e)
        {
            //Faz o cast do objeto sender para a caixa de texto
            TextBox tb = (TextBox)sender;

            //Testa se a informação é válida e configura a cor de fundo
            //de acordo com o resultado
            if (tb.Text.Length == 0 && tb != txtOcupacao)
            {
                tb.Tag = false;
                tb.BackColor = Color.Red;
            }
            else if (tb == txtOcupacao && (tb.Text.Length != 0 && tb.Text.CompareTo("Programador") != 0))
            {
                //Não muda as cores aqui, pois as cores irão mudar enquanto
                //o usuário está digitando
                tb.Tag = false;
            }
            else
            {
                tb.Tag = true;
                tb.BackColor = SystemColors.Window;
            }

            //Chama ValidarTudo para configurar o botão OK
            ValidarTudo();
        }
    }

```

Desta vez, devemos descobrir exatamente qual o controle que está chamando o manipulador de eventos, pois não queremos que a cor de fundo para a caixa de texto *Ocupação* seja alterada para vermelho assim que o usuário começar a sua digitação. Fazemos isto verificando a propriedade **Nome** da caixa de texto que nos foi passada no parâmetro **sender**.

Apenas uma coisa permanece: o método **ValidarTudo()** que ativa ou desativa o botão **OK**.

Digitar o seguinte:

```

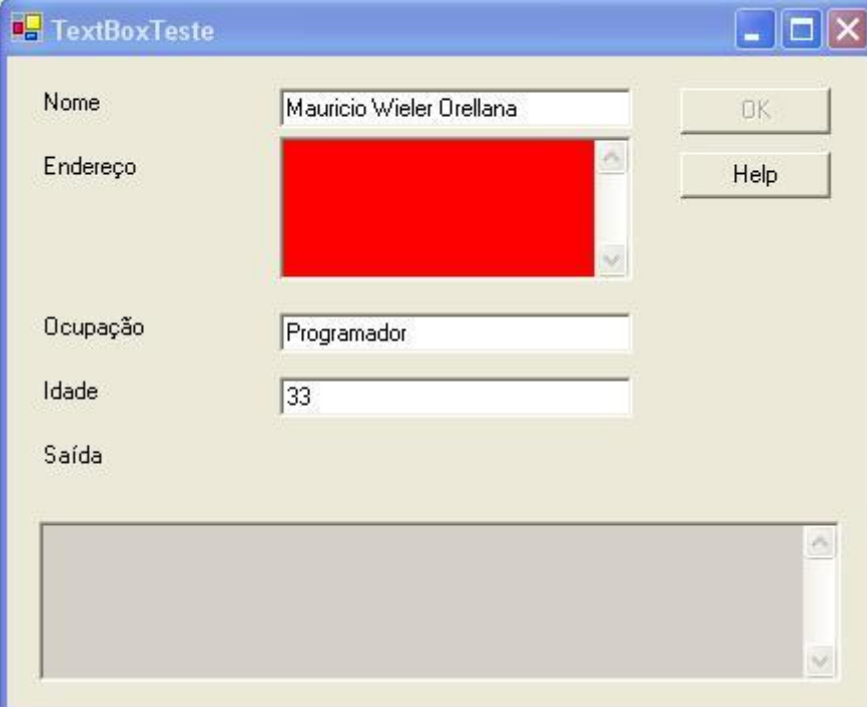
private void ValidarTudo()
{
    //Define o botão OK para ativado se todas as tags forem
    true
    this.btnOK.Enabled = ((bool)(this.txtEndereco.Tag) &&
        (bool)(this.txtIdade.Tag) && (bool)(this.txtNome.Tag) &&
        (bool)(this.txtOcupacao.Tag));
}

```

Obs. : Esse método vem após o método que foi digitado anteriormente que era **private void txtBox_TextoMudado**

O método simplesmente define o valor da propriedade **Enabled** do botão OK para **true** se todas as propriedades **Tag** forem **true**. Precisamos classificar um valor para essas propriedades **Tag** como booleanos porque ele é armazenado como um tipo objeto.

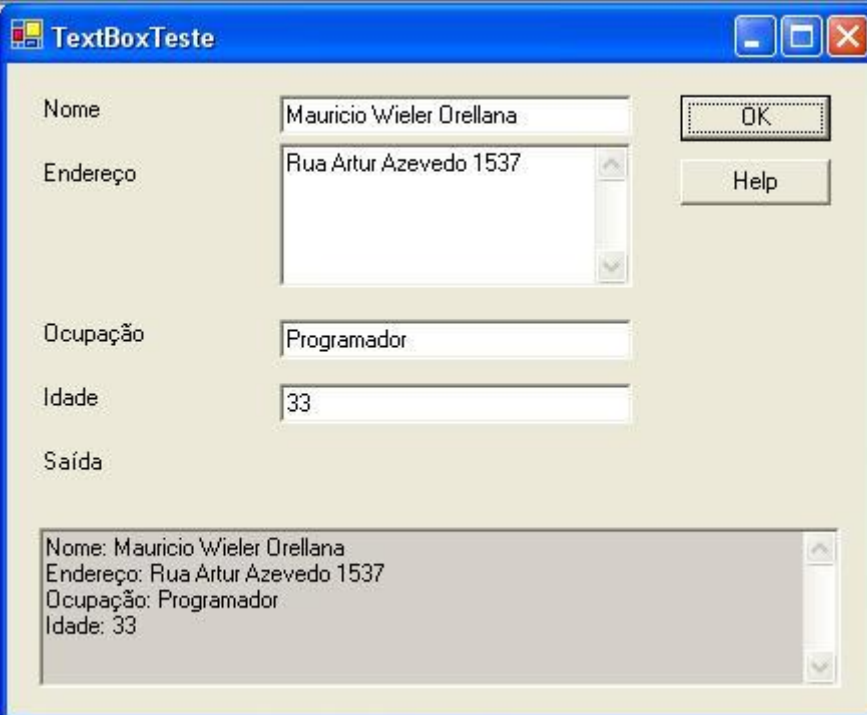
Se testar o programa, agora, você deverá ver uma tela como a da figura abaixo:



The screenshot shows a Windows Form titled "TextBoxTeste". It contains the following elements:

- Nome:** Text box containing "Mauricio Wieler Orellana".
- Endereço:** Text box containing a redacted address (represented by a solid red box).
- Ocupação:** Text box containing "Programador".
- Idade:** Text box containing "33".
- Saída:** A large, empty text box at the bottom.
- Buttons:** "OK" and "Help" buttons are located to the right of the form fields.

Ou, finalmente:



The screenshot shows the same Windows Form "TextBoxTeste" but with valid data entered:

- Nome:** "Mauricio Wieler Orellana".
- Endereço:** "Rua Artur Azevedo 1537".
- Ocupação:** "Programador".
- Idade:** "33".
- Saída:** The large text box now displays the entered data: "Nome: Mauricio Wieler Orellana", "Endereço: Rua Artur Azevedo 1537", "Ocupação: Programador", and "Idade: 33".
- Buttons:** "OK" and "Help" buttons are still present.

Note que você pode dar um clique no botão **Help** enquanto está em uma caixa de texto com dados inválidos sem que a cor de fundo seja alterada para vermelho.

Iremos continuar neste assunto, com maiores detalhes neste editor e com aplicações MDI e SDI.

Eventos e delegações

Criar eventos para objetos no C# é muito fácil graças a uma tecnologia chamada *delegação* (***delegations***). A única coisa importante para entender que os capacitará a criar eventos é compreender que deve fazer assinatura do evento. Fazer a assinatura de um evento significa fornecer o código que será executado quando um evento é gerado no formato idêntico ao manipulador de eventos.

Na verdade, os próprios manipuladores de eventos são simplesmente funções.

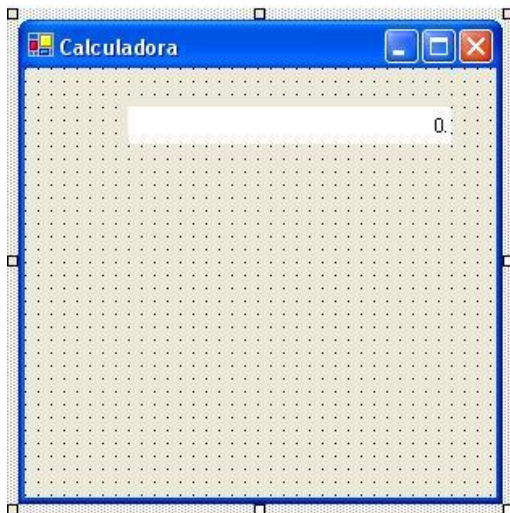
Definitivamente, a única restrição em um manipulador de eventos é que ele deve coincidir com a assinatura (tipo de retorno e parâmetros) obrigatória do evento. Essa assinatura faz parte da definição de um evento e é especificada por uma delegação.

Falando em termos bem gerais, fazendo uma analogia, delegações seriam algo como tipos por referencia em funções, ao invés de conter valores para variáveis, contem um endereço de uma função.

Criaremos, para entender esse conceito melhor, uma calculadora funcional, onde todos os botões e seus eventos *Click* serão criados dinamicamente.

Projeto Calculadora

Crie um novo projeto do tipo Windows Application em C# e coloque apenas um componente ***TextBox*** conforme sugere a imagem a seguir:



Para facilitar nosso código, vamos chamar o componente ***TextBox*** de ***lblDisplay*** e, para ficar com a aparência da imagem acima, altere a propriedade ***BorderStyle*** para ***None***. Na propriedade *Text*, coloque “0.”.

Seguindo a boa técnica de programação e de Engenharia de Software, vamos, inicialmente, declarar as variáveis que serão utilizadas neste programa.

Precisamos armazenar os valores que sofrerão as operações matemáticas e necessitamos também de alguma inteligência para definir os valores dos botões. Para este tipo de problema é muito interessante usar enumerações.

O código com as declarações de variáveis, fica assim:

```
public class Form1 : System.Windows.Forms.Form
{
    // Variáveis para armazenar os valores
    private float Value1 = 0;
    private float Value2 = 0;
    // Variáveis enumerações para operadores e números
    private enum OperatorFlag {None, Plus, Minus, Multiply, Divide} ;
    private enum KeyType {None, Number, Cancel, Operator, Equal} ;
    private OperatorFlag OperatorType ;
}
```

O próximo passo é criar a rotina que vai gerar, denominar e mostrar os botões no **form**. A sugestão para esta rotina é:

```
private void CreateButtons()
{
    // Loop para criar os botões com números
    int j;
    for (int i=0;i<9;i++)
    {
        j = (int)(i/3);
        this.AddButton(25 + (i%3)*50, 50+ ((2-j)*50), (i+1).ToString(),KeyType.Number);
    }

    // Adiciona os demais operadores e especiais
    this.AddButton(25, 200, "0",KeyType.Number);
    this.AddButton(75,200,"C",KeyType.Cancel);
    this.AddButton(125,200,"=",KeyType.Equal);
    this.AddButton(175,50,"+",KeyType.Operator);
    this.AddButton(175,100,"-",KeyType.Operator);
    this.AddButton(175,150,"*",KeyType.Operator);
    this.AddButton(175,200,"/",KeyType.Operator);
}
```

Como observação, cabe ressaltar que o código anterior parece confuso, mas se trata apenas de um algoritmo matemático para a distribuição dos botões (suas posições no **Form**). É exatamente isso que estamos fazendo no primeiro **this.AddButton();**

Se repararem com atenção, o método **AddButton** recebe como parâmetros os seguintes valores: Posição à esquerda, posição ao topo, texto e tipo de operador. Foi pensando nessa função que foi criado, anteriormente, o enumerador **KeyType**.

Se vocês compilarem agora, aparecerá um erro, o que é obvio, porque o método **AddButton** não faz parte das bibliotecas .NET, é um método que devemos implementar. Se vocês notarem, ainda não criamos os botões.

Vamos, então, à implementação do método em questão, digitando no **Code Editor** do Visual Studio.NET:

```
.....  
this.AddButton(175,150,"*",KeyType.Operator);  
this.AddButton(175,200,"/",KeyType.Operator);  
}  
  
private void AddButton(int Left, int Top, string Key, KeyType Type)  
{  
    //Cria o botão e o adiciona ao form  
    Button newButton = new Button();  
    this.Controls.Add(newButton);  
  
    //Designa Valores para o novo botão  
    newButton.Top = Top;  
    newButton.Left = Left;  
    newButton.Width = 35;  
    newButton.Height = 35;  
    newButton.Text = Key;  
}
```

Agora sim, você pode compilar e executar seu programa. A aparência dele deve ser igual à da imagem a seguir:



Visualmente, está tudo perfeito, porem, ainda não funciona nada. É exatamente aí que entra a magia das delegações. Vamos criar os eventos que serão disparados a partir do clique do mouse em cima do botão.

Novamente, o enumerador **KeyType** vai funcionar bem, ele vai determinar qual o tipo de evento que nosso botão deve disparar. Veja o código que deve ser digitado no método **AddButtons**, logo abaixo dos códigos que já havíamos digitado:


```
newButton.Height = 35;
newButton.Text = Key;
//Associa um evento ao botão
switch(Type)
{
    case KeyType.Number:
        newButton.Click += new System.EventHandler(NumberClick);
        break;
    case KeyType.Operator:
        newButton.Click += new System.EventHandler(OperatorClick);
        break;
    case KeyType.Equal:
        newButton.Click += new System.EventHandler(EqualsClick);
        break;
    case KeyType.Cancel:
        newButton.Click += new System.EventHandler(CancelClick);
        break;
}
```

Como você pode perceber, o objeto que estamos usando para gerar eventos é uma instancia da classe **System.EventHandler**. Agora que vamos escrever a função que será disparada pelo evento, como **NumberClick** e outras três, devemos nos preocupar com que a assinatura da função seja idêntica à delegação padrão definida do .NET Framework para o evento **Click**. O código, fica assim:

```
        break;
    case KeyType.Cancel:
        newButton.Click += new System.EventHandler(CancelClick);
        break;
    }
}

public void NumberClick(Object sender, System.EventArgs e)
{
    string Key = ((Button)sender).Text;

    Valuel= Valuel * 10 + int.Parse(Key);
    lblDisplay.Text = Valuel.ToString();
}

public void CancelClick(Object sender, System.EventArgs e)
{
    Valuel = 0;
    Value2 = 0;
    lblDisplay.Text = " 0";
}
```

```
public void OperatorClick(Object sender, System.EventArgs e)
{
    string Key = ((System.Windows.Forms.Button)sender).Text;

    switch(Key)
    {
        case "+":
            OperatorType = OperatorFlag.Plus;
            break;
        case "-":
            OperatorType = OperatorFlag.Minus;
            break;
        case "*":
            OperatorType = OperatorFlag.Multiply;
            break;
        case "/":
            OperatorType = OperatorFlag.Divide;
            break;
    }
    Value2= Value1;
    Value1= 0;
}

public void EqualsClick(Object sender, System.EventArgs e)
{
    switch(OperatorType)
    {
        case OperatorFlag.Plus:
            Value1 = Value2 + Value1;
            break;
        case OperatorFlag.Minus:
            Value1 = Value2 - Value1;
            break;
        case OperatorFlag.Multiply:
            Value1 = Value2 * Value1;
            break;
        case OperatorFlag.Divide:
            Value1 = Value2 / Value1;
            break;
    }
    OperatorType = OperatorFlag.None;
    lblDisplay.Text = Value1.ToString();
    Value2 = 0;
}
```

A lógica utilizada nesta calculadora é a mais simples possível. Sem dúvida, poderíamos ter criado algo mais avançado, mas não faria sentido ao nosso exemplo sobre criar componentes e eventos dinamicamente.

Analizando o código anterior, note que são acumulados valores na variável **Value1** até que um operador matemático seja clicado. Neste momento, armazenamos o **OperatorType** e os próximos valores são lançados na variável **Value2**.

Ao clicar em "=", pegamos o operador armazenado em **OperatorType** e o aplicamos em **Value1** e **Value2**, devolvendo para o **TextBox** chamado **lblDisplay** o resultado da operação matemática.

Para que o código funcione bem, não se esqueça de chamar o método que cria os botões no Formulário principal, que é o **CreateButtons()**.

Digite no código gerado no Visual Studio.NET :

```
public Form1()
{
    //
    // Required for Windows Form Designer support
    //
    InitializeComponent();
    CreateButtons();

    //
    // TODO: Add any constructor code after InitializeComponent call
    //
}
```

Com isso, terminamos o projeto chamado *Calculadora* e podemos passar o código completo, a parte.