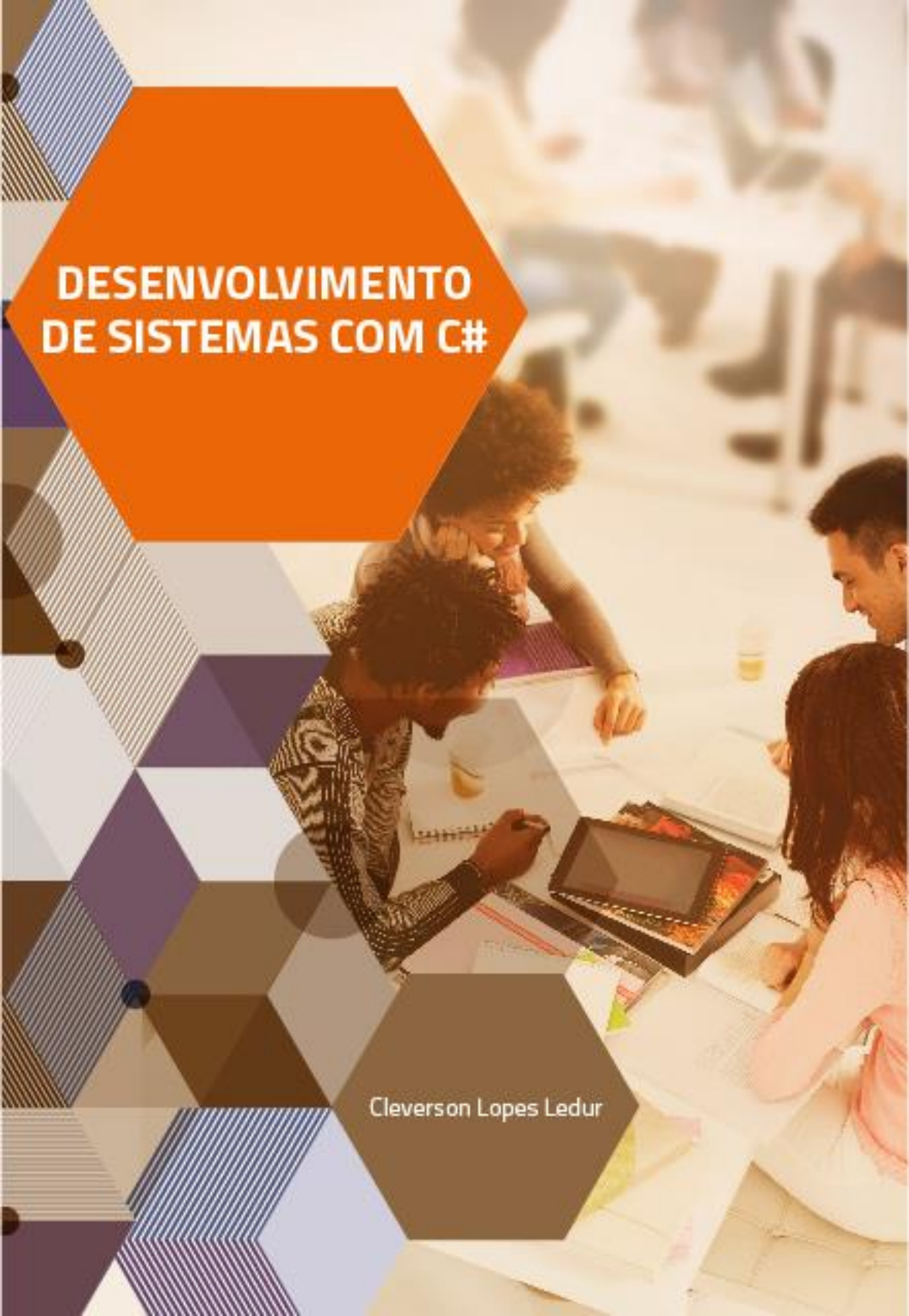
DESENVOLVIMENTO DE SISTEMAS COM C#



Cleverson Lopes Ledur

Revisão técnica:

Jeferson Faleiro Leon
Desenvolvimento de Sistemas
Especialista Formação Pedagógica de Professores
Professor do curso Técnico em informática

L475d Ledur, Cleverson Lopes.
Desenvolvimento de sistemas com #C [recurso eletrônico] / Cleverson Lopes Ledur; [revisão técnica: Jeferson Faleiro Leon]. – Porto Alegre : SAGAH, 2018.

ISBN 978-85-9502-314-7

1. Ciência da computação. 2. Linguagens de programação de computador. I. Título.

CDU 004.43

Catalogação na publicação: Karin Lorien Menoncin CRB-10/2147

Utilizar o Windows Forms para o desenvolvimento de aplicações visuais – II

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Alterar as propriedades e eventos em uma aplicação Windows Forms.
- Reconhecer os principais eventos e propriedades de um formulário.
- Utilizar os principais eventos e propriedades de um botão.

Introdução

Um evento é uma ação a que você pode responder ou que pode "manipular" no código. Aplicativos dirigidos por eventos executam o código em resposta a um evento. Os tipos de eventos criados por um objeto variam, mas muitos tipos são comuns à maioria dos controles. Logo, em uma aplicação Windows Forms, é muito importante conhecer os eventos e também as propriedades que pode utilizar a fim de obter uma boa interface ao final do projeto. De um lado, você pode manipular as ações com os eventos. De outro, as propriedades dão a possibilidade de personalizar e modificar os elementos durante a execução de um programa em C#.

Neste capítulo, você vai aprender a alterar as propriedades e eventos em uma aplicação Windows Forms. Também vai entender os principais eventos e propriedades de um formulário (form) e de um botão (button).

Eventos e propriedades

Um evento é uma ação a que você pode responder ou que pode "manipular" no código. Os eventos podem ser gerados por uma ação do usuário, como clicar no mouse ou pressionar uma tecla. Eles também podem ser criados pelo código do programa ou pelo sistema. Aplicativos dirigidos por eventos executam o código em resposta a um evento. Cada formulário e controle expõe um conjunto predefinido de eventos que você pode programar. Se um desses eventos ocorre e existe um código no manipulador de eventos associado, esse código é invocado (MICROSOFT, c2018).

Os tipos de eventos criados por um objeto variam, mas muitos tipos são comuns à maioria dos controles. Por exemplo, a maioria dos objetos irá lidar com um evento **Click**. Se um usuário clicar em um formulário, o código no manipulador de eventos **Click** do formulário será executado (DEITEL; DEITEL, 2016).

Um manipulador de eventos é um método vinculado a um evento. Quando o evento é gerado, o código dentro do manipulador de eventos é executado. Cada manipulador de eventos fornece dois parâmetros que permitem manipular o evento corretamente. No código abaixo você pode ver um manipulador de eventos para um evento de clique do controle de botão.

```
private void button1_Click(object sender, System.EventArgs e)
{

}
```

O primeiro parâmetro, `sender`, fornece uma referência ao objeto que elevou o evento. O segundo parâmetro, `e`, o exemplo anterior, passa um objeto específico para o evento que está sendo tratado. Ao referenciar as propriedades do objeto (e, às vezes, seus métodos), você pode obter informações como a localização do mouse para eventos do mouse, ou dados sendo transferidos nos eventos de arrastar e soltar.

Saiba mais

Normalmente, cada evento produz um manipulador de eventos com um tipo de evento-objeto diferente para o segundo parâmetro. Alguns manipuladores de eventos, como os dos eventos MouseDown e MouseUp, têm o mesmo tipo de objeto para o segundo parâmetro. Nesse caso, você pode usar o mesmo manipulador de eventos para lidar com ambos os eventos.

Você também pode usar o mesmo manipulador de eventos para lidar com o mesmo evento para diferentes controles. Por exemplo, se você tiver um grupo de controles RadioButton em um formulário, você poderia criar um único manipulador de eventos para o evento Click e ter o evento Click de cada controle vinculado ao manipulador de evento único.

No Visual Studio, você pode visualizar os eventos disponíveis para o item selecionado no menu **Propriedades**, como mostra a Figura 1. Você deve verificar se o ícone com um raio está selecionado.

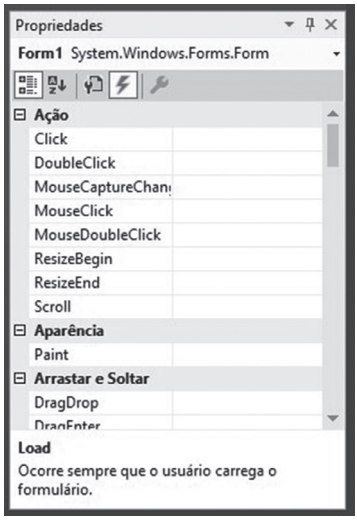


Figura 1. O ícone com o raio deve estar selecionado.

Nessa tela, se você clicar duas vezes em uma ação, por exemplo, **Click**, o método já será criado e você será redirecionado para ele a fim de inserir o código da ação, conforme a Figura 2.

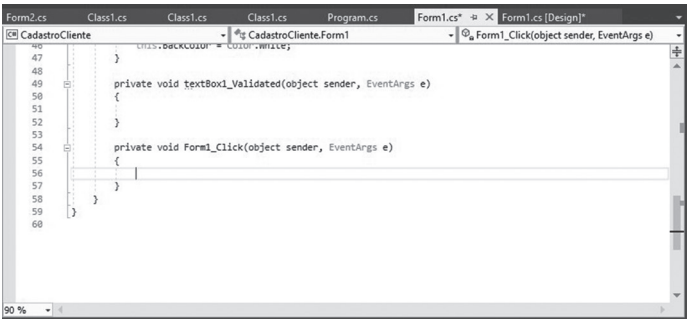


Figura 2. Inserção do código da ação.

Agora, suponha que você quer que, ao clicar no formulário, a cor de fundo dele seja alterada. A cor do formulário nada mais é do que uma propriedade. As propriedades são alguns valores que descrevem o componente e que podem ser alterados. Por exemplo, a cor de fundo é composta pela propriedade **BackColor**. Como você está trabalhando dentro da classe do componente, pode chamá-la com `this.BackColor`.

As propriedades de um componente também são acessadas por meio do menu **Propriedades**. Observe a Figura 3. Você deve verificar se o ícone com uma folha e uma ferramenta está ativo.

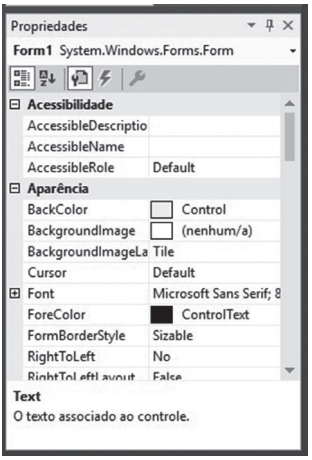


Figura 3. Icone com uma folha e uma ferramenta ativada.

Saiba mais

Existem diversos tipos de propriedades. Entre elas, você pode considerar:

- **Acessibilidade:** relacionada com a acessibilidade do componente.
- **Aparência:** relacionada com aspectos visuais do elemento, como cores, formatos, etc.
- **Comportamento:** propriedade voltada para habilitação ou não de determinados comportamentos do elemento.
- **Dados:** compostos por decisões de informações do componente e inserção de dados.
- **Design:** voltado para elementos do projeto, como localização, linguagem, etc.
- **Diversos:** elementos não classificados em outras opções.
- **Estilo de janela:** propriedades que descrevem como a janela deve ser apresentada.
- **Foco:** voltado para questões de validação do componente.
- **Layout:** propriedades relacionadas com tamanho, posição e características do componente.

Propriedades e eventos de um Form

Um form é um dos componentes principais do Windows Forms, já que é nele que você vai colocar todos os outros componentes. Ele pode ser entendido como uma janela do seu sistema. Em um sistema Windows Forms, você pode ter diversos forms diferentes, cada um com suas propriedades e eventos específicos. Veja, na Figura 4, uma ilustração do componente form.

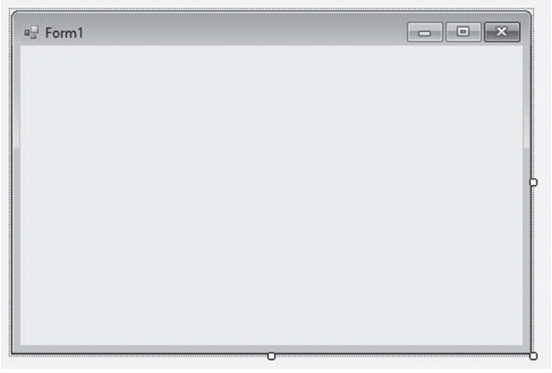


Figura 4. Componente form.

Nesta seção, você vai conhecer os principais eventos e propriedades que um componente form pode possuir. Com esse conhecimento, você já pode criar, modificar e implementar novos sistemas.

Propriedades

Um componente form possui um conjunto de propriedades que permitem a personalização de acordo com as necessidades do sistema. Veja a seguir as principais propriedades encontradas em um form.

- AcceptButton: obtém ou define o botão do formulário que é clicado quando o usuário pressiona a tecla **ENTER**.
- AccessibleDescription: obtém ou define a descrição do controle usado por aplicativos de cliente de acessibilidade.
- AccessibleName: obtém ou define o nome do controle usado pelos aplicativos do cliente de acessibilidade.
- Container: obtém o `IContainer` que contém o componente (herdado do componente).
- ContainsFocus: obtém um valor indicando se o controle, ou um de seus controles filhos, atualmente possui o foco de entrada.
- ContextMenu: obtém ou define o menu de atalho associado ao controle.
- ContextMenuStrip: obtém ou define o `ContextMenuStrip` associado a esse controle.
- ControlBox: obtém ou define um valor indicando se uma caixa de controle é exibida na barra de legenda do formulário.
- Controls: obtém a coleção de controles contidos no controle.
- Criado: obtém um valor indicando se o controle foi criado.
- CreateParams: obtém os parâmetros de criação necessários quando o identificador de controle é criado.
- Focused: obtém um valor indicando se o controle tem foco de entrada.
- Font: obtém ou define a fonte do texto exibida pelo controle.
- FontHeight: obtém ou define a altura da fonte do controle.
- ForeColor: obtém ou define a cor do primeiro plano do controle.
- FormBorderStyle: obtém ou define o estilo de borda do formulário.
- Handle: obtém o identificador de janela ao qual o controle está vinculado.
- HasChildren: obtém um valor indicando se o controle contém um ou mais controles filhos.
- Heigth: obtém ou define a altura do controle.

- HelpButton: obtém ou define um valor indicando se um botão de ajuda deve ser exibido na caixa de legenda do formulário.
- HorizontalScroll: obtém as características associadas à barra de rolagem horizontal (herdado de `ScrollableControl`).
- HScroll: obtém ou define um valor que indica se a barra de rolagem horizontal está visível.
- Icon: obtém ou define o ícone para o formulário.
- ImeMode: obtém ou define o modo Input Method Editor (IME) do controle.
- ImeModeBase: obtém ou define o modo IME de um controle.
- InvokeRequired: obtém um valor indicando se o chamador deve utilizar um método de invocação ao fazer chamadas de método para o controle porque o chamador está em um segmento diferente do que o controle no qual foi criado.
- IsAccessible: obtém ou define um valor que indica se o controle está visível para aplicativos de acessibilidade.
- IsDisposed: obtém um valor indicando se o controle foi descartado.
- IsHandleCreated: obtém um valor indicando se o controle possui um identificador associado a ele.
- IsMdiChild: obtém um valor indicando se o formulário é um formulário filho de interface de documento múltiplo (MDI).
- IsMdiContainer: obtém ou define um valor indicando se o formulário é um contêiner para formulários secundários de interface de documentos múltiplos (MDI).
- IsMirrored: obtém um valor indicando se o controle é espelhado.
- IsRestrictedWindow: essa API suporta a infraestrutura do produto e não se destina a ser usada diretamente do seu código. Obtém um valor indicando se o formulário pode usar todos os eventos de entrada do Windows e do usuário sem restrições.
- KeyPreview: obtém ou define um valor indicando se o formulário receberá eventos-chave antes de o evento ser passado para o controle que tenha foco.
- LayoutEngine: obtém uma instância em cache do mecanismo de layout do controle.
- Left: obtém ou define a distância, em pixels, entre a margem esquerda do controle e a borda esquerda da área do cliente do seu recipiente.
- Location: obtém ou define o ponto que representa o canto superior esquerdo do formulário nas coordenadas da tela.
- MainMenuStrip: obtém ou define o recipiente de menu principal para o formulário.
- Margin: obtém ou define o espaço entre controles.

- MaximizeBox: obtém ou define um valor indicando se o botão **Maximizar** é exibido na barra de legenda do formulário.
- MaximizedBounds: obtém e define o tamanho do formulário quando ele é maximizado.
- MaximumSize: obtém o tamanho máximo ao qual o formulário pode ser redimensionado (substitui `Control.MaximumSize`).
- MdiChildren: obtém uma série de formulários que representam os formulários secundários de interface de documentos múltiplos (MDI) que são compartilhados com esse formulário.
- MdiParent: obtém ou define o formulário pai atual da interface de documentos múltiplos (MDI) desse formulário.
- Menu: obtém ou define o `MainMenu` que é exibido no formulário.
- MergedMenu: obtém o menu mesclado para o formulário.
- MinimizeBox: obtém ou define um valor indicando se o botão **Minimizar** é exibido na barra de legenda do formulário.
- MinimumSize: obtém ou define o tamanho mínimo ao qual o formulário pode ser redimensionado (substitui `Control.MinimumSize`).
- Modal: obtém um valor indicando se esse formulário é exibido de forma modal.

Eventos

Um componente form possui um conjunto de eventos que permitem alterar o comportamento do sistema conforme ações são realizadas sobre ele. Veja a seguir os principais eventos que você pode encontrar em um form.

- Activate: ocorre quando o formulário é ativado no código ou pelo usuário.
- AutoSizeChanged: ocorre quando a propriedade **AutoSize** muda.
- AutoValidateChanged: ocorre quando a propriedade **AutoValidate** muda.
- BackColorChanged: ocorre quando o valor da propriedade **BackColor** muda.
- BackgroundImageChanged: ocorre quando o valor da propriedade **BackgroundImage** muda.
- BackgroundImageLayoutChanged: ocorre quando a propriedade **BackgroundImageLayout** muda.
- BindingContextChanged: ocorre quando o valor da propriedade **BindingContext** muda.

- CausesValidationChanged: ocorre quando o valor da propriedade **CausesValidation** muda.
- ChangeUICues: ocorre quando o foco ou a interface do usuário do teclado (UI) se alteram.
- Click: ocorre quando o controle é clicado.
- ClientSizeChanged: ocorre quando o valor da propriedade **ClientSize** muda.
- Close: Ocorre quando o formulário está fechado.
- Closing: ocorre quando o formulário está fechando.
- ContextMenuChanged: ocorre quando o valor da propriedade **ContextMenu** muda.
- ContextMenuStripChanged: ocorre quando o valor da propriedade **ContextMenuStrip** muda.
- ControlAdded: ocorre quando um novo controle é adicionado ao **Control.ControlCollection**.
- ControlRemoved: ocorre quando um controle é removido do **Control.ControlCollection**.
- CursorChanged: ocorre quando o valor da propriedade **Cursor** muda.
- Deactivate: ocorre quando o formulário perde o foco e não é mais o formulário ativo.

Propriedades e eventos de um button

Já imaginou um sistema sem botões? Ou uma janela de informação sem um botão **OK** ou **Cancelar** para você tomar uma decisão? Botões são um dos componentes mais utilizados na programação visual com Windows Forms. Para criar botões, você precisa utilizar o componente button, que possui um conjunto bastante amplo de propriedades e eventos. Veja, na Figura 5, uma ilustração de um componente button no Visual Studio.

Figura 5. Componente button.

Agora você vai ver um passo a passo para alterar a propriedade e um evento de um componente do tipo button.

Primeiro, você deve criar um projeto do tipo Windows Forms (Figura 6). Então, adicione um botão no formulário.

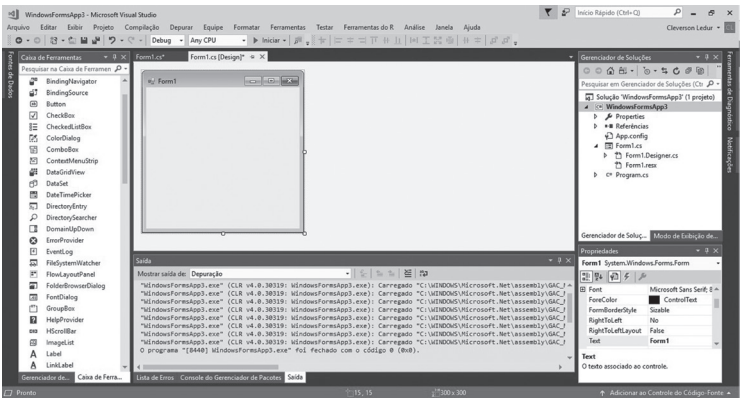


Figura 6. Formulário.

Veja, na Figura 7, que o botão está com o texto **button1**. Primeiramente, então, você precisa fazer a alteração de uma propriedade chamada **text** para alterar esse valor.

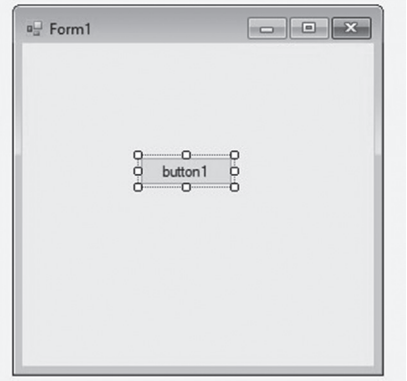


Figura 7. Criação de um botão.

No menu **Propriedades**, você vai encontrar inicialmente a propriedade text. Clique duas vezes sobre o campo de valor e digite qualquer texto. No exemplo da Figura 8, foi utilizado o texto **Ok**.

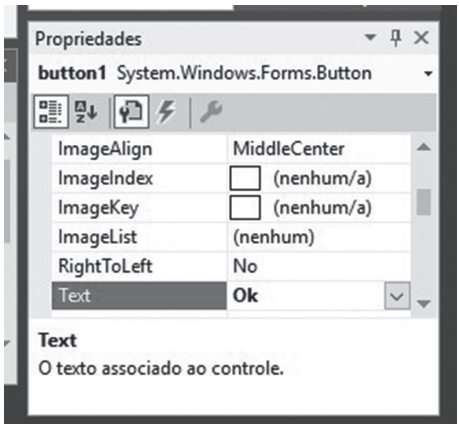


Figura 8. Digite qualquer texto no campo do valor da propriedade.

Agora, você pode visualizar, na Figura 9, a representação visual do formulário que mostra o texto do botão já alterado para **Ok**.

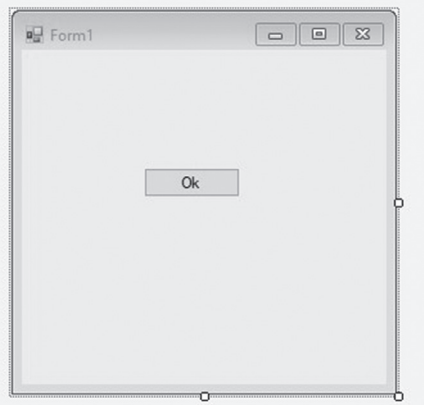


Figura 9. Texto do botão alterado.

Você vai agora fazer uma alteração no evento do componente. Para isso, vai mudar o modo de comportamento para o evento `Click`. No menu **Propriedades**, clique no ícone em formato de raio. Clique então duas vezes no campo em branco do evento `Click`, conforme a Figura 10.

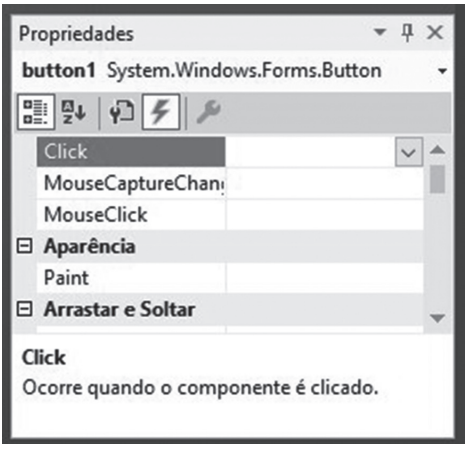


Figura 10. Modificando o modo de comportamento.

O editor de códigos será aberto. Veja que automaticamente é criado um método dentro da classe do form. Nele, você pode inserir o código que será executado quando o usuário clicar no botão. Observe a Figura 11.

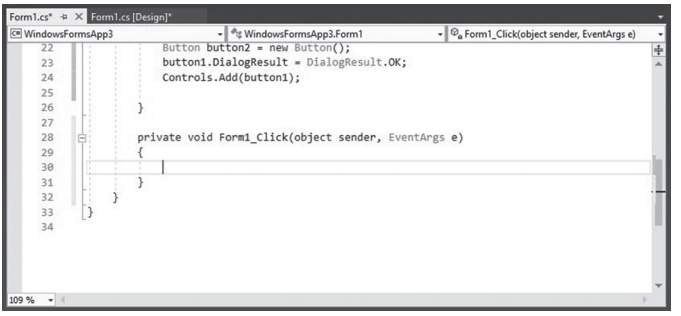


Figura 11. Editor de códigos aberto.

Como exemplo, você pode inserir um código para exibir um alerta (mensagem) assim que o usuário clicar no botão. Dentro do alerta será exibida a mensagem **Você clicou em Ok!**. Veja o próximo exemplo.

```
private void Form1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Você clicou em Ok!");
}
```

Propriedades

Um componente button possui um conjunto de propriedades que permitem a personalização de acordo com as necessidades do sistema. Veja a seguir as principais propriedades encontradas em um button.

- AcessibilidadeObjeto: obtém o `AccessibleObject` atribuído ao controle.
- AccessibleDefaultActionDescription: obtém ou define a descrição de ação padrão do controle para uso por aplicativos de cliente de acessibilidade.
- AccessibleDescription: obtém ou define a descrição do controle usado por aplicativos de cliente de acessibilidade.
- AccessibleName: obtém ou define o nome do controle usado pelos aplicativos do cliente de acessibilidade.
- AccessibleRole: obtém ou define o papel acessível do controle (herdado do controle).
- AllowDrop: obtém ou define um valor que indica se o controle pode aceitar dados que o usuário arrasta para ele.
- Anchor: obtém ou define as bordas do recipiente ao qual um controle é vinculado e determina como um controle é redimensionado com seu pai.
- AutoEllipsis: obtém ou define um valor indicando se o caractere de reticências (...) aparece na borda direita do controle, denotando que o texto de controle se estende além do comprimento especificado do controle (herdado de `ButtonBase`).
- AutoScrollOffset: obtém ou define em que esse controle é roteado em `ScrollControlIntoView`.
- AutoSize: obtém ou define um valor que indica se o controle é redimensionado com base em seus conteúdos (herdado de `ButtonBase`).
- AutoSizeMode: obtém ou define o modo pelo qual o botão redimensiona automaticamente.

- BackColor: obtém ou define a cor de fundo do controle (herdado de `ButtonBase`).
- BackgroundImage: obtém ou define a imagem de fundo exibida no controle.
- BackgroundImageLayout: obtém ou define o layout da imagem de plano de fundo conforme definido na enumeração `ImageLayout`.
- BindingContext: obtém ou define o `BindingContext` para o controle.
- Bottom: obtém a distância, em pixels, entre a borda inferior do controle e a borda superior da área de cliente do seu recipiente.
- Bounds: obtém ou define o tamanho e a localização do controle, incluindo seus elementos não clientes, em pixels, em relação ao controle pai.
- CanEnableIme: obtém um valor indicando se a propriedade ImeMode pode ser definida como um valor ativo, para habilitar o suporte IME.
- CanFocus: obtém um valor indicando se o controle pode receber foco.
- CanRaiseEvents: determina se os eventos podem ser criados no controle.
- CanSelect: obtém um valor indicando se o controle pode ser selecionado.
- Capture: obtém ou define um valor indicando se o controle capturou o mouse.
- CausesValidation: obtém ou define um valor que indica se o controle faz com que a validação seja realizada em qualquer controle que exija validação quando recebe foco.
- ClientRectangle: obtém o retângulo que representa a área do cliente do controle.
- ClientSize: obtém ou define a altura e a largura da área do cliente do controle.
- CompanyName: obtém o nome da empresa ou criador do aplicativo que contém o controle.
- Container: obtém o `IContainer` que contém o componente (herdado do componente).
- ContainsFocus: obtém um valor indicando se o controle, ou um de seus controles filhos, atualmente possui o foco de entrada.
- ContextMenu: obtém ou define o menu de atalho associado ao controle.
- ContextMenuStrip: obtém ou define o `ContextMenuStrip` associado a esse controle.
- Controls: obtém a coleção de controles contidos no controle.
- Criado: obtém um valor indicando se o controle foi criado.

- CreateParams: obtém um `CreateParams` na classe base ao criar uma janela.
- Cursor: obtém ou define o cursor que é exibido quando o ponteiro do mouse está sobre o controle.

Eventos

- AutoSizeChanged: ocorre quando o valor da propriedade **AutoSize** muda (herdado de `ButtonBase`).
- BackColorChanged: ocorre quando o valor da propriedade **BackColor** muda.
- BackgroundImageChanged: ocorre quando o valor da propriedade **BackgroundImage** muda.
- BackgroundImageLayoutChanged: ocorre quando a propriedade **BackgroundImageLayout** muda.
- BindingContextChanged: ocorre quando o valor da propriedade **BindingContext** muda.
- CausesValidationChanged: ocorre quando o valor da propriedade **CausesValidation** muda.
- ChangeUICues: ocorre quando o foco ou a interface do usuário do teclado (UI) se alteram.
- Click: ocorre quando o controle é clicado.
- ClientSizeChanged: ocorre quando o valor da propriedade **ClientSize** muda.
- ContextMenuChanged: ocorre quando o valor da propriedade **ContextMenu** muda.
- ContextMenuStripChanged: ocorre quando o valor da propriedade **ContextMenuStrip** muda.
- ControlAdded: ocorre quando um novo controle é adicionado ao `Control.ControlCollection`.
- ControlRemoved: ocorre quando um controle é removido do `Control.ControlCollection`.
- CursorChanged: ocorre quando o valor da propriedade **Cursor** muda.
- Disposed: ocorre quando o componente é descartado por uma chamada para o método `Dispose` (herdado do componente).
- DockChanged: ocorre quando o valor da propriedade **Dock** muda.
- DoubleClick: ocorre quando o usuário clica duas vezes no controle do botão.

- DpiChangedAfterParent: ocorre quando a configuração do DPI para um controle é modificada programaticamente após o DPI do seu controle ou formulário original ter mudado.
- DpiChangedBeforeParent: ocorre quando a configuração do DPI para um controle é alterada de forma programática antes que um evento de alteração de DPI para o controle ou o formulário pai tenha ocorrido.
- DragDrop: ocorre quando uma operação de arrastar e soltar é concluída.
- DragEnter: ocorre quando um objeto é arrastado para os limites do controle.
- DragLeave: ocorre quando um objeto é arrastado para fora dos limites do controle.
- DragOver: ocorre quando um objeto é arrastado pelos limites do controle.
- EnabledChanged: ocorre quando o valor da propriedade **Enabled** foi alterado.
- Enter: ocorre quando o controle é inserido.
- FontChanged: ocorre quando o valor da propriedade **Font** é alterado.
- ForeColorChanged: ocorre quando o valor da propriedade **ForeColor** muda.
- GiveFeedback: ocorre durante uma operação de arrastar.
- GotFocus: ocorre quando o controle recebe foco.
- HandleCreated: ocorre quando um identificador é criado para o controle.
- HandleDestroyed: ocorre quando o identificador do controle está em processo de destruição.
- HelpRequested: ocorre quando o usuário solicita ajuda para um controle.
- ImeModeChanged: essa API suporta a infraestrutura do produto e não se destina a ser usada diretamente do seu código. Ocorre quando a propriedade **ImeMode** é alterada. Esse evento não é relevante para essa classe (herdado de **ButtonBase**).
- Invalidated: ocorre quando a exibição de um controle requer redessificação.
- KeyDown: ocorre quando uma tecla é pressionada enquanto o controle está focado.
- KeyPress: ocorre quando um personagem A tecla espaço, ou quando retrocesso é pressionada enquanto o controle está focado.
- KeyUp: ocorre quando uma chave é liberada enquanto o controle tem foco.

- Layout: ocorre quando um controle deve reposicionar seus controles filhos.
- Leave: ocorre quando o foco de entrada deixa o controle.
- LocationChanged: ocorre quando o valor da propriedade **Location** mudou.

Referências

DEITEL, P.; DEITEL, H. *Visual C#:* how to program. 6th ed. London: Pearson, 2016.

MICROSOFT. *System.Windows.Forms Namespace.* Redmond, c2018. Disponível em: <https://msdn.microsoft.com/en-us/library/system.windows.forms(v=vs.110).aspx>. Acesso em: 02 nov. 2017.

Leitura recomendada

SLACK, N. et al. *Gerenciamento de operações e de processos:* princípios e práticas de impacto estratégico. 2. ed. Porto Alegre: Bookman, 2013.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo: