Windows Forms(2)

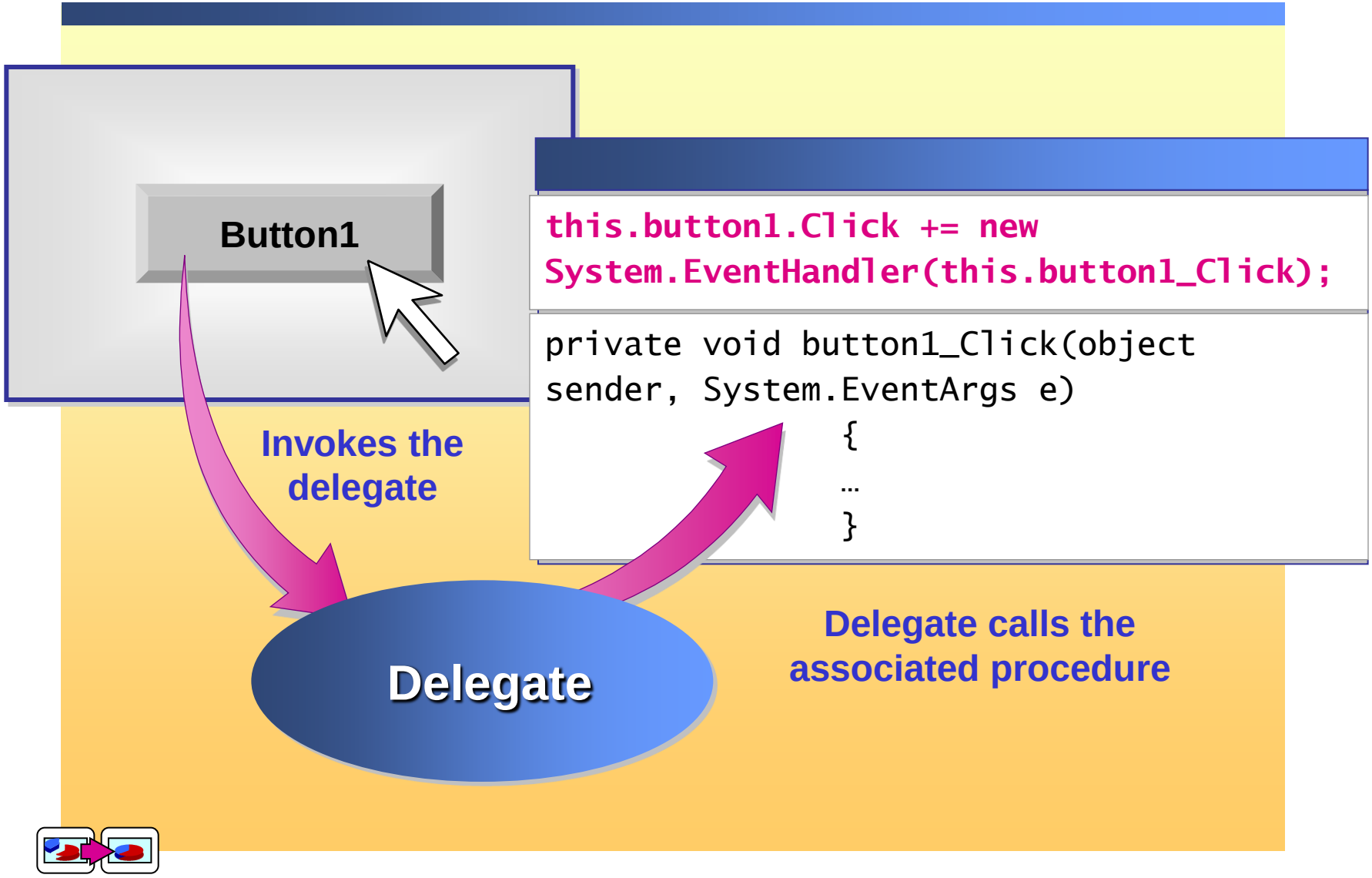
Programiranje korisničkih interfejsa

Bojan Furlan

Lesson: Creating an Event Handler for a Control

- Event Model in the .NET Framework
- What Are Delegates?
- What Is an Event Handler?
- How to Create Event Handlers for Control Events
- How to Add and Remove Event Handlers at Run Time
- Practice: Creating an Event Handler for a Control

Event Model in the .NET Framework



What Are Delegates?

- **Delegate**
  - Binds events to methods
  - Can be bound to single or multiple methods
- **When an event is recorded by an application**
  - The control raises the event by invoking the delegate for the event
  - The delegate in turn calls the bound method

```
public delegate void AlarmEventHandler(object
   sender, AlarmEventArgs e);
```

What Is an Event Handler?

- **Event Handlers**
  - Methods bound to an event
  - When the event is raised, the code within the event handler is executed
- **Two Event Arguments with Event Handlers**
  - An object representing the object that raised the event
  - An event object containing any event-specific information

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

}
```

How to Create Event Handlers for Control Events

```
// inside the Windows Form Designer generated code region
…
this.exitButton.Click +=
       new System.EventHandler(this.exitButton_Click);

// add the exitToolStripMenuItem.click  event to
  exitButton_click handler

this.exitToolStripMenuItem.Click += new
  System.EventHandler(this.exitButton_Click);

private void exitButton_Click(object sender,
  System.EventArgs e)
{
       this.Close();

}
```

How to Add and Remove Event Handlers at Run Time

- To associate an event with an event handler at run time, use the AddHandler statement

```
this.button2.Click += new
      System.EventHandler(this.button1_Click);
```

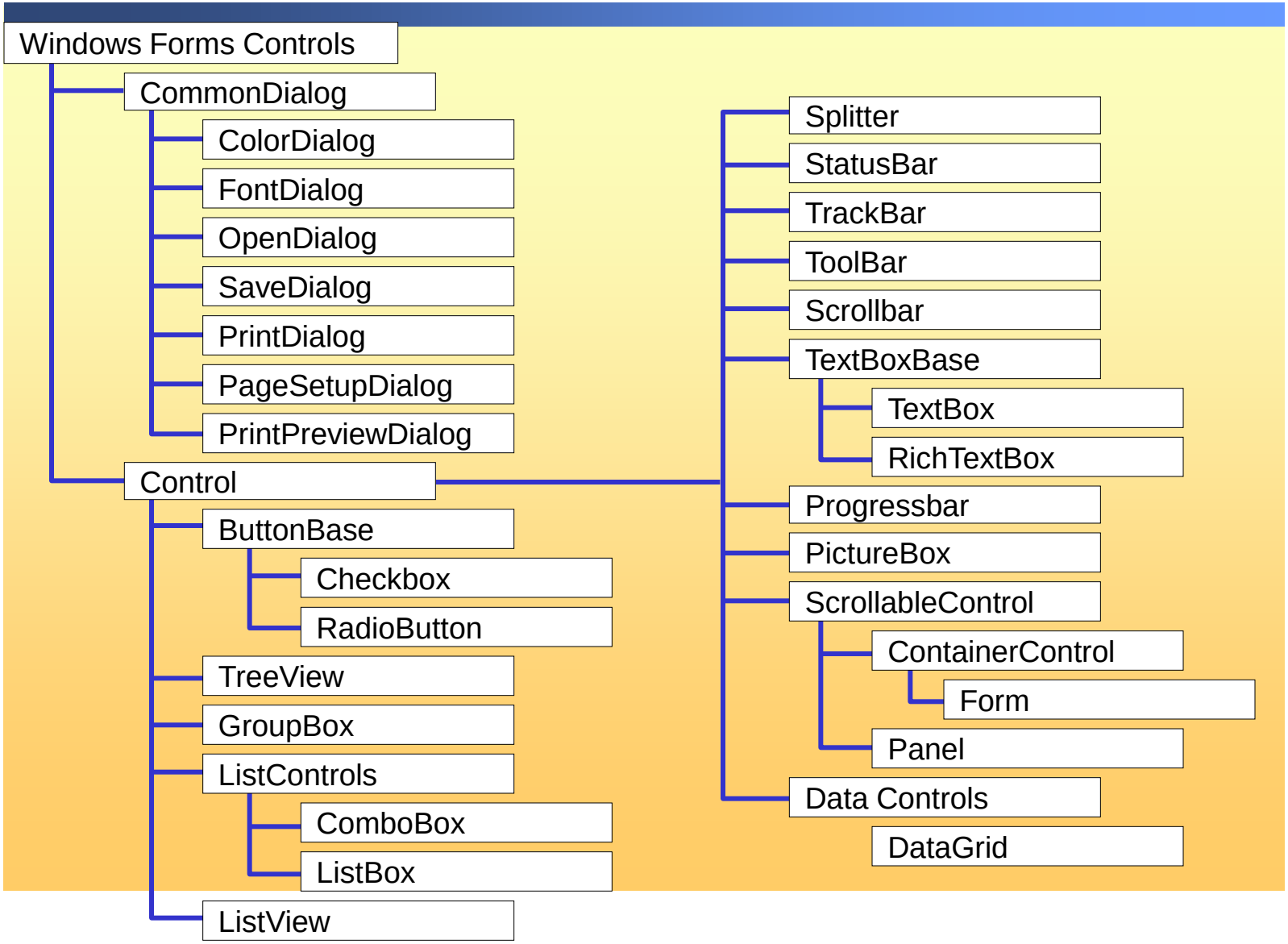
- To remove the association of an event with an event handler at run time, use the RemoveHandler statement

```
this.button2.Click -= new
System.EventHandler(this.button1_Click);
```

Lesson: Using Windows Forms Controls

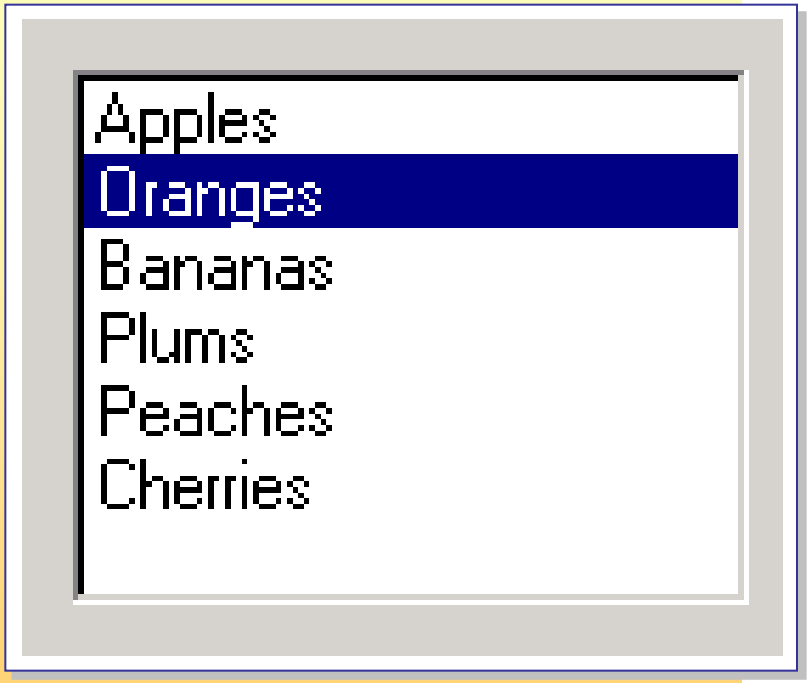
- Selecting a Windows Forms Control Based on Function
- How to Use the ListBox Control
- How to Use Container Controls

Selecting a Windows Forms Control Based on Function



How to Use the ListBox Control

1. Add a **ListBox** control to the form
2. Add items to the **ListBox** using the **Items** collection
3. Configure the properties of the **ListBox** control




```
ListBox1.Items.Addrange(NewObject()
   {“Apples”,“Oranges”,“Bananas”});
```

How to Use Container Controls

1. Drag a container (**GroupBox, Panel, TabControl, TableLayoutPanel** …) control from the Toolbox onto a form
2. Add other controls to the container control, drawing each inside the panel
3. If you have existing controls that you want to enclose in the container, drag them into the container
4. To display scroll bars for the **Panel** control, set its **AutoScrollbar** property to **True**
5. To display a caption on the **GroupBox**, set its **Text** property to an appropriate caption

Selecting Dialog Boxes in Visual Studio .NET

OpenFileDialog	Allows users to open files through a preconfigured dialog box
SaveFileDialog	Selects files to save and the location where they are saved
ColorDialog	Allows users to select a color from the palette and add colors to it
FontDialog	Exposes the fonts that are currently installed on the system
PrintDialog	Selects a printer and other printer-related settings
PageSetupDialog	Sets up page details for printing
PrintPreviewDialog	Displays a document as it would appear when it is printed

How to Display Dialog Boxes in an Application

- To display a preconfigured Visual Studio .NET dialog box

```
private void button1_Click(object sender,
  System.EventArgs e)
{
   OpenFileDialog1.ShowDialog();
}
```

- To display a message dialog box

```
public void PerformCalculations()
{
   MessageBox.Show ("The search is now complete",
  "My Application",
MessageBoxButtons.OKCancel,
  MessageBoxIcon.Asterisk);
}
```

DialogResult Property

DialogResult Property

- Use the value returned by this property to determine what action the user has taken

Example

The value **DialogResult.Cancel** indicates that user clicked the **Cancel** button

- **DialogResult** property can be set at design time or run time

How to Use Input from Dialog Boxes

To retrieve and use results from dialog boxes

1. In the Code Editor, navigate to the event handler or the method for which you want to set the **DialogResult** property
2. Add code to retrieve **DialogResult** value

```
public void DisplayValue()
{
  DialogResult userResponse  = openFileDialog1.ShowDialog();
  if (userResponse == DialogResult.OK)
  {
       filePath = openFileDialog1.FileName.ToString();
       MessageBox.Show("You successfully opened: '" + filePath +
  "'", "Success", MessageBoxButtons.OK,
  MessageBoxIcon.Information, MessageBoxDefaultButton.Button1);
  }
…
}
```

Demonstration: Using the Dialog Control

In this demonstration, you will see how to

- Add an **Dialog** control to your project
- Create the code to display the **Dialog**
- Set the **Dialog** properties

Lesson: Adding Controls at Run Time

- Controls Collection
- How to Add Controls at Run Time

Controls Collection

- **Controls Collection**
  - Represents a collection of Control objects
  - Use the **Add**, **Remove**, and **RemoveAt** methods to add and remove controls from the collection

```
Form1.Controls.Add(textbox1);
```

```
Form1.Controls.Remove(textbox1);
```

  - Use the **Contains** method to determine whether or not a control is a part of the collection

```
Form1.Controls.Contains(textbox1);
```

Controls Collection

ControlCollection Methods

The following table lists some of the methods of **ControlCollection**.

Method	Description
Add	Adds the specified control to the control collection.
AddRange	Adds an array of control objects to the collection.
Clear	Removes all controls from the collection.
Contains	Determines whether the specified control is a member of the collection.
Remove	Removes the specified control from the control collection.
RemoveAt	Removes a control from the control collection at the specified indexed location.
ToString (inherited from Object)	Returns a String that represents the current Object.
IndexOf	Retrieves the index of the specified control in the control collection.
GetEnumerator	Returns an enumerator that can be used to iterate through the control collection.

How to Add Controls at Run Time

To add controls at run time

1. Create the control that will be added to your container

```
CheckBox signatureCheckBox = new CheckBox();
// set properties
signatureCheckBox.Text = "Signature required";
signatureCheckBox.Left = 24;
signatureCheckBox.Top = 80;
```

2. Add the control to the container using the **Add** method of the **Controls** property

```
// add the new control to the collection
GroupBox1.Controls.Add(signatureCheckBox);
```

Lesson: Creating Menus

- How to Add a Context Menu to a Form
- How to Add Menu Items at Run Time
- How to Create Event Handlers for Menu Items
- How to Use Menu Properties

How to Add a Context Menu to a Form

To add controls at run time

1. In the Toolbox, double-click the **ContextMenu** control
2. Associate the context menu with a form or a control by setting that object's **ContextMenu** property

To add a context menu programmatically

```
public void AddContextMenu()
{
 ContextMenu mnuContextMenu = new ContextMenu();
 this.ContextMenu = mnuContextMenu;

}
```

How to Add Menu Items at Run Time

To add menu items to a context menu at run time

1. Within the method, create **MenuItem** objects to add to the Context Menu Object collection

```
MenuItem menuItemNew = new MenuItem();
```

2. Within the method, set the **Text** property for each menu item

```
menuItemNew.Text = "&New";
```

3. Within the method, add menu items to the **MenuItems** collection of the **ContextMenu** object

```
contxMenu.MenuItems.Add(menuItemNew);
```

How to Create Event Handlers for Menu Items

To add functionality to menu items

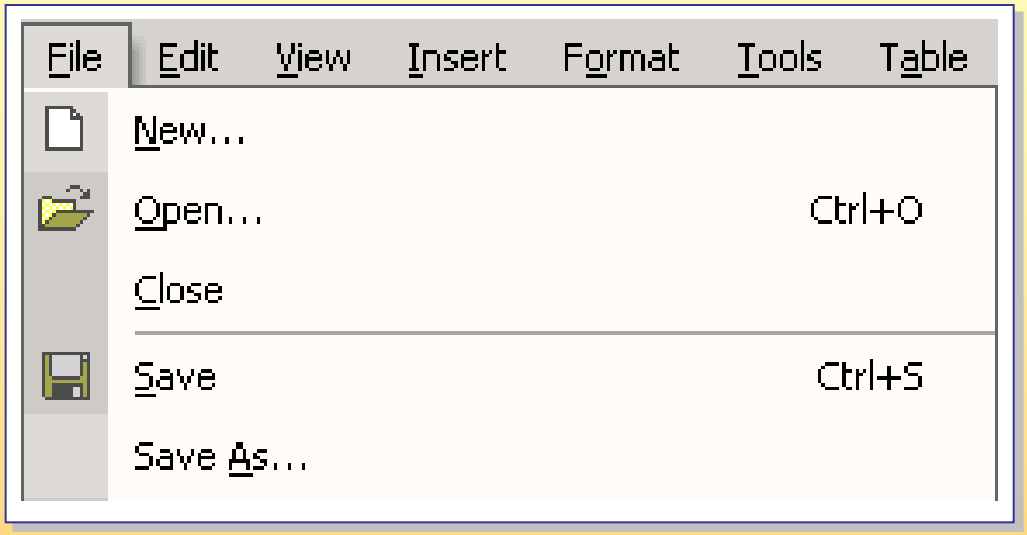
1. Create an event handler for the **MenuItem.Click** event

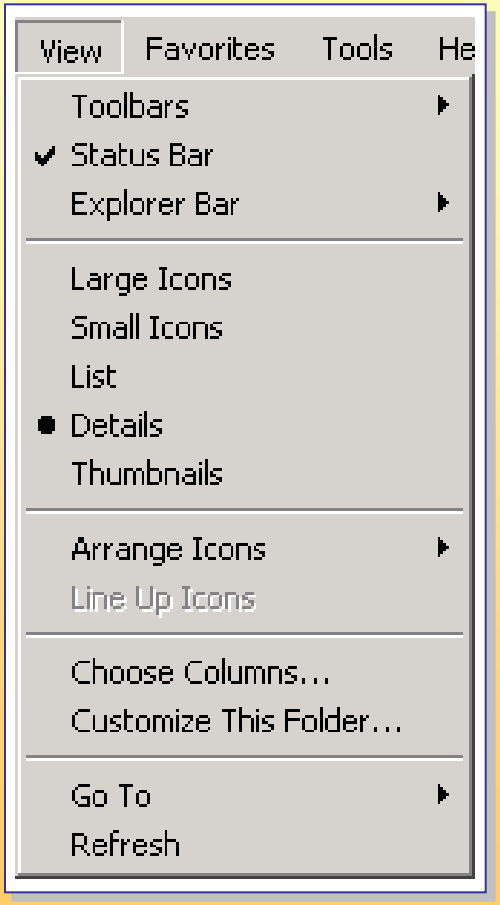
```
private void menuItem1_Click(object sender,
   System.EventArgs e)

       {

        }
```

2. Write the code to handle the event

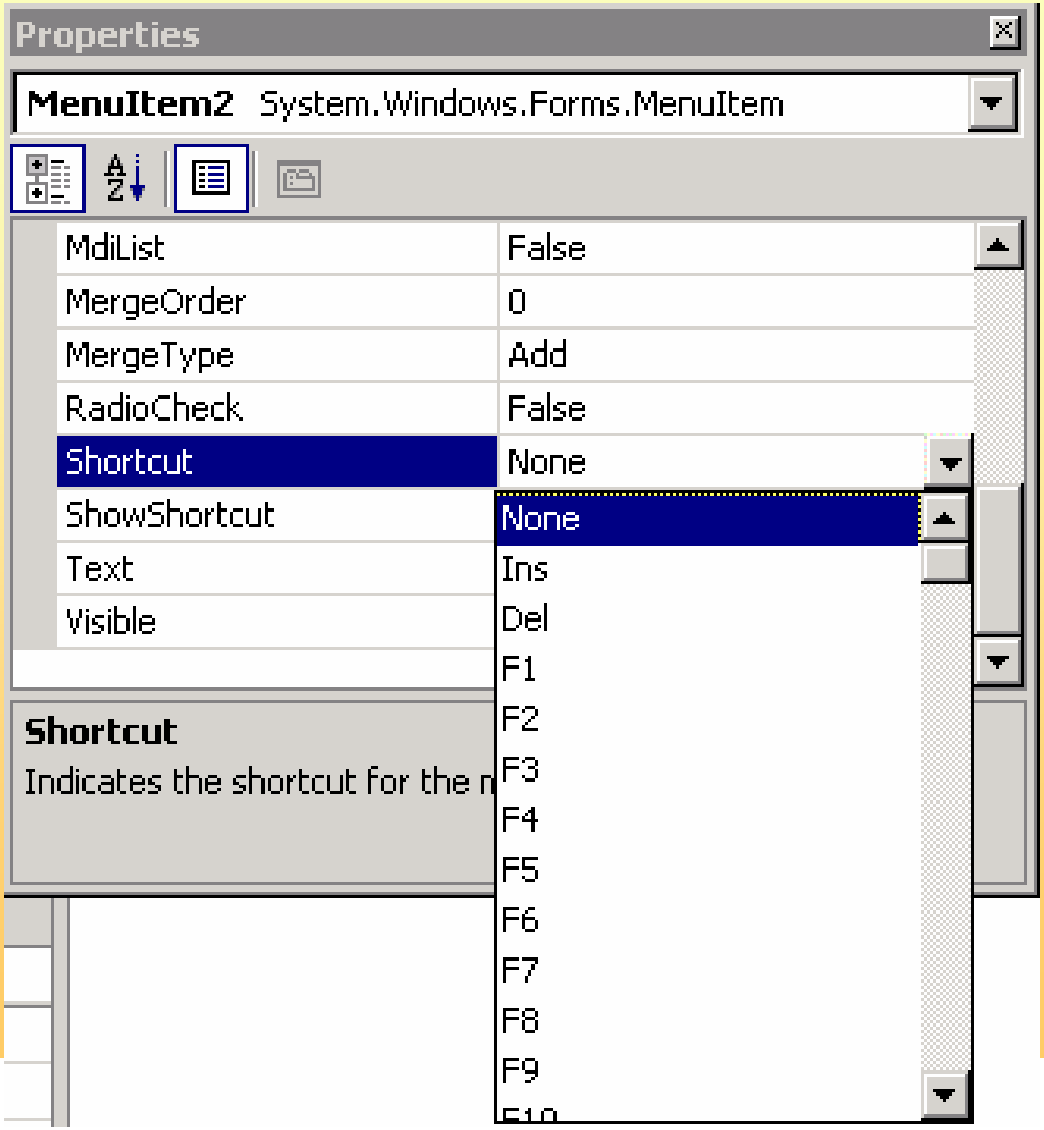
```
private void menuItem1_Click(object sender,
   System.EventArgs e)

       {

        MessageBox.Show("You clicked the File
       menu.","The Event Information");

        }
```

How to Use Menu Properties

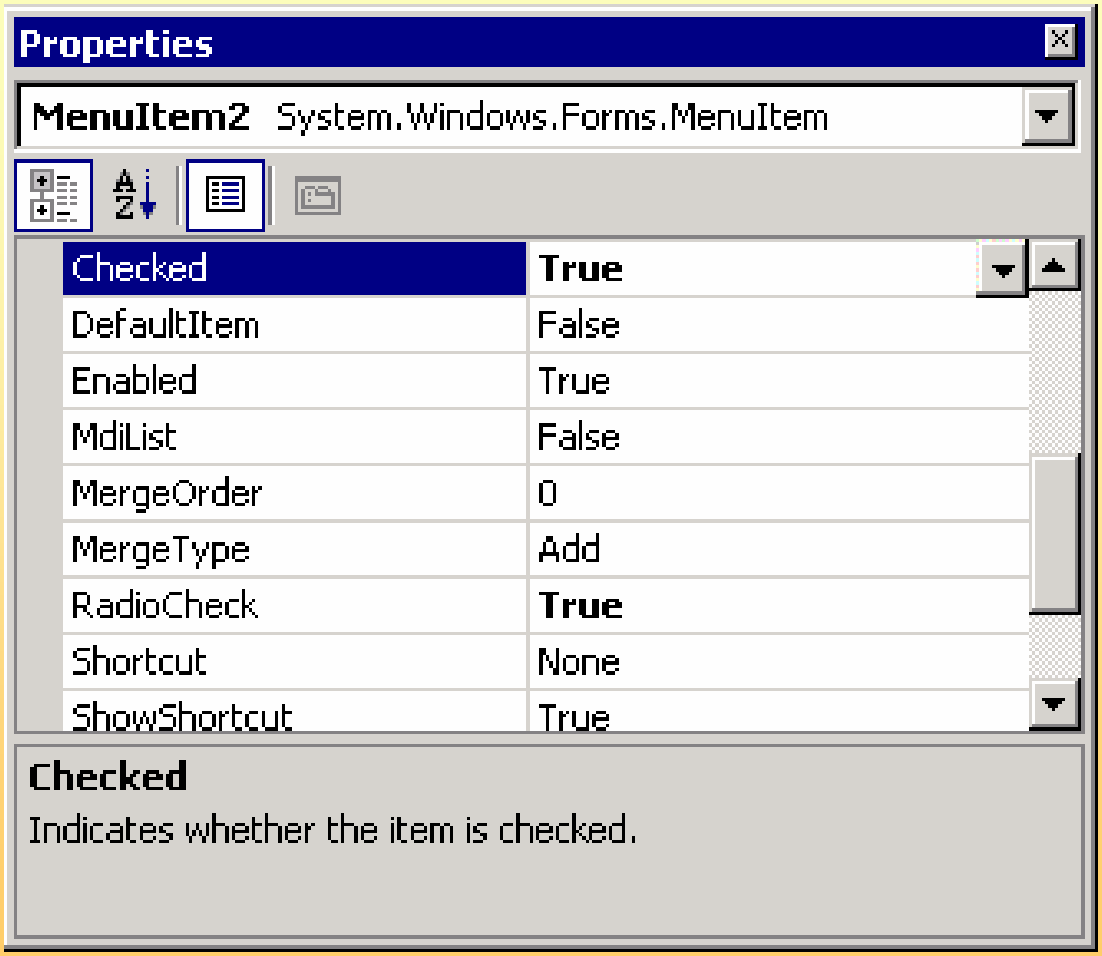




Shortcut keys



Displaying checked menu items



Lesson: Validating User Input

- How to Validate Controls by Using the Validating Event
- ErrorProvider Control
- How to Use the ErrorProvider Control
- Demonstration: Validating Data in a Windows Forms Application

Validation

- **Validating event occurs before a control loses focus**
- **The Validated event fires after the validation of the controls finishes running the validating events**
- **The CausesValidation property determines whether the previous control will participate in validation.**
  - If set to False for a control, the previous control does not fire the validation event (e.g. Cancel button)

How to Validate Controls by Using the Validating Event

- Use the Validating event of a control to validate user input

```
private void minValueTextBox_Validating(object sender,
System.ComponentModel.CancelEventArgs e)
{
    if (Convert.ToInt32(minValueTextBox.Text) >=
    Convert.ToInt32(maxValueTextBox.Text))
        {
            e.Cancel = true; // cancel shifting focus
            MessageBox.Show("You must enter a minimum
   value " +"that is less than the maximum value");
        }
}
```

ErrorProvider Control

- **ErrorProvider**
  - Displays errors when validating user input on a form
  - Displays errors within a dataset
- **Key Properties**

  DataSource | ContainerControl | Icon
- **Key Method**

  SetError

How to Use the ErrorProvider Control

To use the ErrorProvider control

1. Add controls to the form
2. Add the ErrorProvider control
2. Add code to the Validating event of the first control

CodeExample

How to Use the ErrorProvider Control

```
protected void textBox1_Validating (object
   sender,
CancelEventArgs e)
{
    try
    {
        int x = Int32.Parse(textBox1.Text);
        …
    }
    catch
    {
        errorProvider1.SetError(textBox1,
        "Not an integer value.");
    }
}
```

Demonstration: Validating Data in a Windows Forms Application

In this demonstration, you will see how to

- Check user keystrokes
- Stop the focus from shifting away from the current control
- Use a message box to provide feedback
- Use an **ErrorProvider** control to provide feedback
- Remove the ErrorProvider icon when the error no longer exists