Event Driven Programming

Elements of Event Driven Programming

Chalew Tesfaye
Department of Computer Science
Debre Berhan University
2017

Objectives

- At the end of the lesson students should be able to:
  - ✓ Understand events and delegates
  - ✓ Create an event handler for a control
  - ✓ Work with windows controls
  - ✓ Work with properties and events of controls
  - ✓ Design and build windows based application

Contents

Elements of Event Driven Programming

- Event Source
  - ✓ Who are the users of the system? Human users, other system, the environment, time, …
- Event
  - ✓ How the user interact to the system? By click, touch, speak, ticking time, sending message, …
- Delegates
  - ✓ Who will receive the events and wire/connect to the event handler?
- Event Handler
  - ✓ Who will define the logic, what will be done when receiving an event?
- User Interface to accept/Listen for the event and respond/display the result
  - ✓ Forms and Controls
  - ✓ Message/Text
  - ✓ Network
  - ✓ Other form of interfaces

Event

- An event is a message sent by an object to signal the occurrence of an action that is either invoked by a user or programmatically.
- Each event has
  - ✓ a sender that raises the event and
  - ✓ a receiver that handles the event.
- An event is used to signal the occurrence of an action.
  - ✓ For example, this action could be user invoked, such as the
    - > Click event of a Button control, or
    - > the event could be raised programmatically to signal the end of a long computation.
- The object that raises (triggers) the event is referred to as the event sender.
- The procedure that handles the event is referred to as the event receiver.
- In either case, the sender does not know which object or method will respond to the events that it raises.

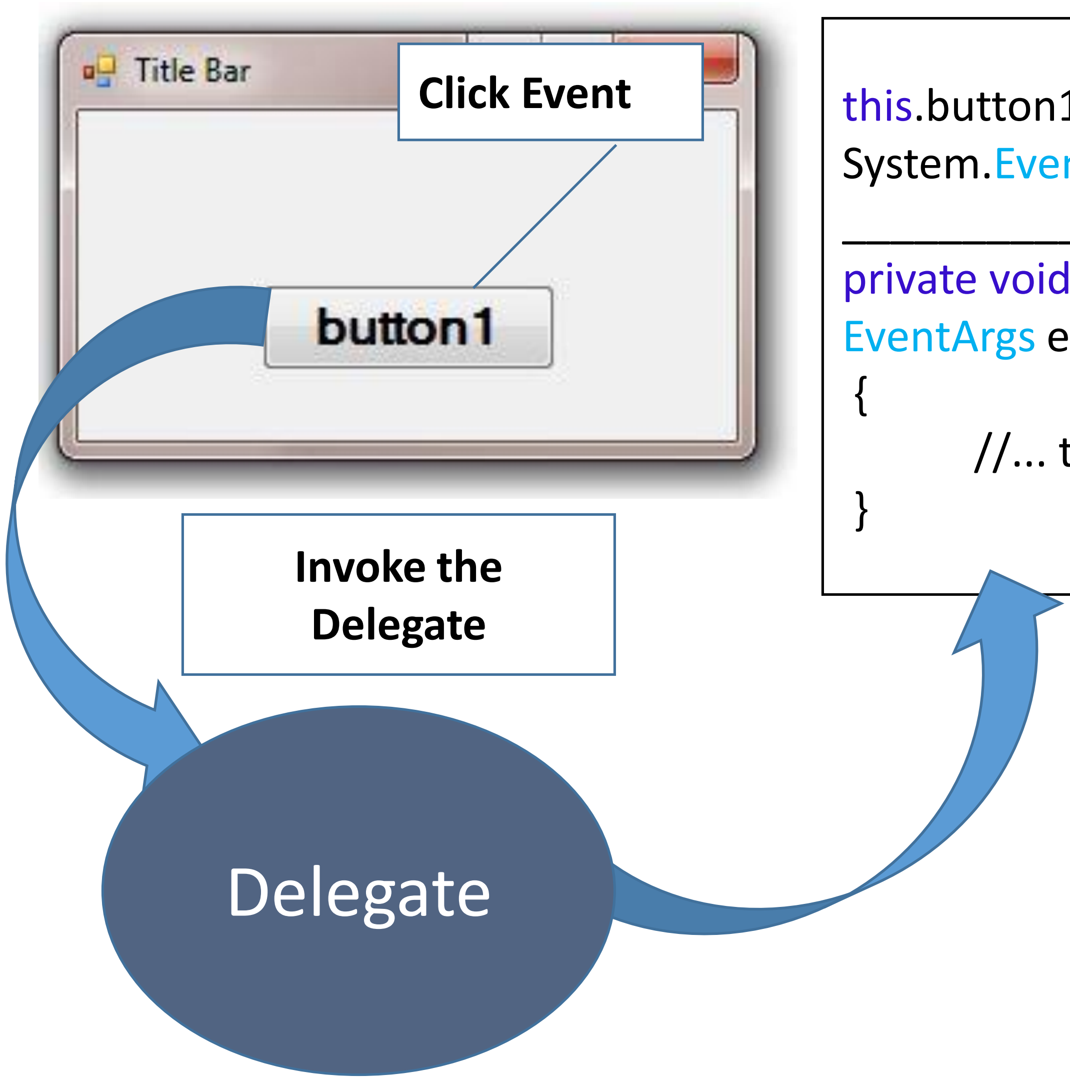
Event …

- Therefore, it is necessary to have a component that links the event sender with the event receiver.
- The .NET Framework uses a delegate type to work as a function pointer between the sender and the event receiver.
- In most cases, the .NET Framework creates the delegate and takes care of the details for you.
- However, you can create your own delegates for cases when you want an event to use different event handlers under different circumstances.

Event …

- Delegates are objects that you can use to call the methods of other objects.
- You can use the Delegate keyword in a declaration statement to create your own delegate that derives from the MulticastDelegate class.
- Creating your own delegates can be useful in situations where you need an intermediary between a calling procedure and the procedure being called.

Event Model in the .NET Framework



```
this.button1.Click += new
System.EventHandler(this.button1_Click);
______________________________________________
private void button1_Click(object sender,
EventArgs e)
 {
         //... the code goes here ...
 }
```

Delegate calls the associated procedures

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 EventHandler(object sender, EventArgs
e);
```

Delegate …

- Delegates are used to hold a reference to the method that will handle an event.
- For example, an event occurs when a user clicks a button.
  - ✓ The button raises a click event but does not know what behavior you, the programmer, want to occur when the button is clicked,
  - ✓ so the button has a delegate member to which you assign your own method for handling the event.
- You can use the same infrastructure that is used by the .NET Framework to create your own delegates.

Delegate …

- A delegate is a data structure, derived from the Delegate Class, which refers
  - ✓ to a static method or
  - ✓ to a class instance and an instance method of that class.
- Delegates are useful when
  - ✓ your application must perform an action by calling a method but
  - ✓ you do not know what that action will be.
- Delegates allow you to specify at run time the method to be invoked.
- Delegates are object-oriented, type-safe, and secure.

Delegate …

- By convention, event delegates in the .NET Framework have two parameters,
  - ✓ the source that raised the event and
  - ✓ the data for the event.
- The following example shows an event delegate declaration:

  ```
  public delegate void EventHandler(object sender, EventArgs e);
  ```

- Event delegates are multicast,
  - ✓ which means that they can hold references to more than one event handling method.
- Delegates allow for flexibility and fine-grain control in event handling.
- A delegate acts as an event dispatcher for the class that raises the event by maintaining a list of registered event handlers for the event.

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 CalculateGpag_Click(object sender, EventArgs e)
{

}
```

Event handler …

- You can use the same event handler to handle more than one event.
  - ✓ For example, you can create a single event handler to handle events of
    - a button and
    - a menu item that are used for the same purpose.
- The following code example is an event handler for the Click event of a button.

```
private void CalculateGpag_Click(object sender, EventArgs e)
{

}
```

Event handler …

- The following code example shows how you can use a single event handler to handle events for multiple controls.

```
// inside the Windows Form Designer generated code region
…
this.button1.Click += new
  System.EventHandler(this.button1_Click);

// add the button2.click event to button1_click handler
this.button2.Click += new
  System.EventHandler(this.button1_Click);

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

Event handler …

- Each event handler provides two parameters that allow you to handle the event properly.
  - ✓ The first parameter (Sender   in the previous code example) provides a reference to the object that raised the event.
  - ✓ It specifies the source that raised the event.
  - ✓ The second parameter (e in the previous code example) passes an object specific to the event being handled.
  - ✓ This parameter contains all of the data that is required to handle the event.

Developing Event Driven App

- Steps to develop a Windows based Event Driven System
  1. Design the GUI
     - Drag/drop controls or using code
     - Set properties of controls at design time/ using code
       - At least some basic properties of Forms and Controls
  2. Select Event(s) for forms and controls
     - Select appropriate events for the forms and controls
  3. Create delegate for the event
     - The IDE does it automatically or you can specify by yourself if you want
  4. Create event handler for controls
     - The IDE does it automatically or you can specify by yourself if you want
  5. Write the event handler
     - Write the business logic code here

Design the GUI

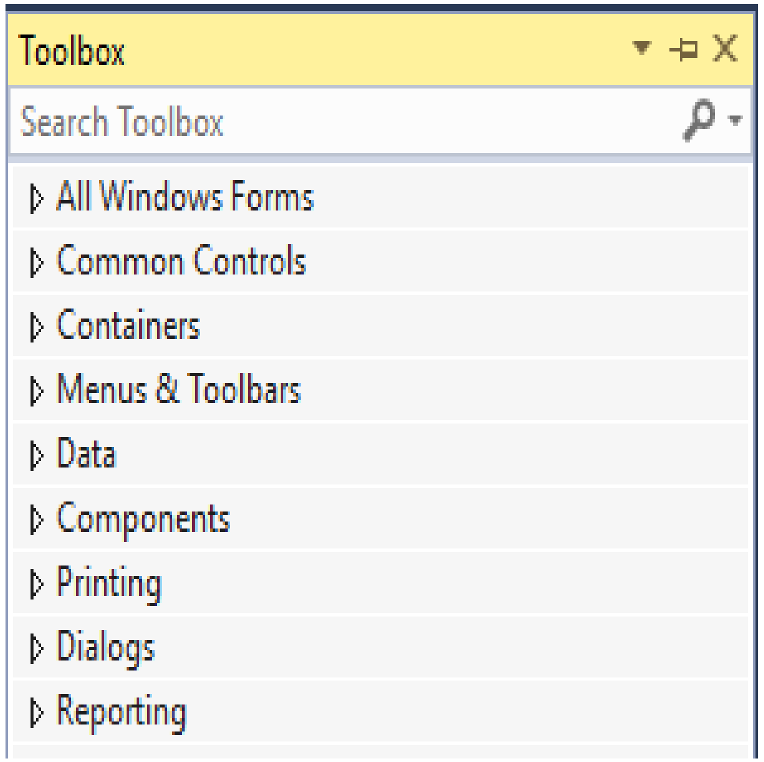
- Windows Form Controls
- Controls
  - ✓ Textbox
  - ✓ Label
  - ✓ Button
  - ✓ RichText Box
  - ✓ RadioButton
  - ✓ Check Box
  - ✓ Link Label
  - ✓ List View
  - ✓ Grid View
  - ✓ …

Windows Form Controls

- Windows Forms controls are reusable components that
  - ✓ encapsulate user interface functionality and
  - ✓ are used in a Windows Forms application.
- Not only does the .NET Framework Class Library
  - ✓ provide many ready-to-use controls,
  - ✓ also provides the infrastructure for developing your own controls.

Windows Form Controls …

- Depending on the functionality that you want to provide in the user interface of your application,
  - ✓ you will select a control from one of the following categories:



- All Windows Forms
- Common Controls
- Containers
- Menus & Toolbars
- Data
- Components
- Printing
- Dialogs
- Reporting

Windows Form Controls …

- Command controls
  - ✓ Button
  - ✓ LinkLabel
  - ✓ NotifyIcon
  - ✓ ToolTip
- Text controls enable users to enter text and edit the text contained in these controls at run time:
  - ✓ Textbox
  - ✓ Rich Textbox
- The following additional text controls can be used to display text but do not allow application users to directly edit the text content that they display:
  - ✓ Label
  - ✓ StatusBar

Windows Form Controls …

- The following selection controls allow users to select a value from a list:
  - ✓ CheckedListBox
  - ✓ ComboBox
  - ✓ DomainUpDown
  - ✓ ListBox
  - ✓ ListView
  - ✓ NumericUpDown
  - ✓ TreeView
- The following are the categories of menu controls:
  - ✓ MenuStrip
  - ✓ ContextMenu

Windows Form Controls …

- Container controls can be used to group other controls on a form
  - ✓ Panel
  - ✓ GroupBox
  - ✓ TabControl
- The following are the categories of graphic controls:
  - ✓ ImageList
  - ✓ PictureBox
- Visual Studio .NET provides a set of common dialog boxes. These include
  - ✓ ColorDialog
  - ✓ FontDialog
  - ✓ PageSetupDialog
  - ✓ PrintDialog
  - ✓ OpenFileDialog
  - ✓ SaveFileDialog
  - ✓ FolderBrowtherDialog

Working with Windows Forms

- Creating a Form
- Adding controls to a Form
- Organizing controls on a Form
- Creating MDI Applications

Windows Forms

- Forms are the basic element of the user interface (UI) in applications created for the Microsoft Windows operating system.
- They provide a framework that you can use throughout your application to give it a consistent **look** and **feel**.
- A form in Windows-based applications is used to present information to the user and to accept input from the user.
- Forms expose properties that define their appearance, methods that define their behavior, and events that define their interaction with the user.

Windows Forms …

- By
  - ✓ setting the properties of the form and
  - ✓ writing code to respond to its events,
  - ✓ customizing the form to meet the requirements of your application.
- A form is a control derived from the Form class,
  - ✓ which in turn derives from the Control class.
- The framework also allows you to inherit from existing forms to add functionality or modify existing behavior.
- When you add a form to your project,
  - ✓ you can choose whether it inherits from the Form class provided by the .NET Framework or from a form you created previously.

How to create a Form

- A base Form is created when you create a new project
- To add another new Form
  - ✓ Right-click the project on the solution explorer
  - ✓ Click Add
  - ✓ Then click Windows Forms

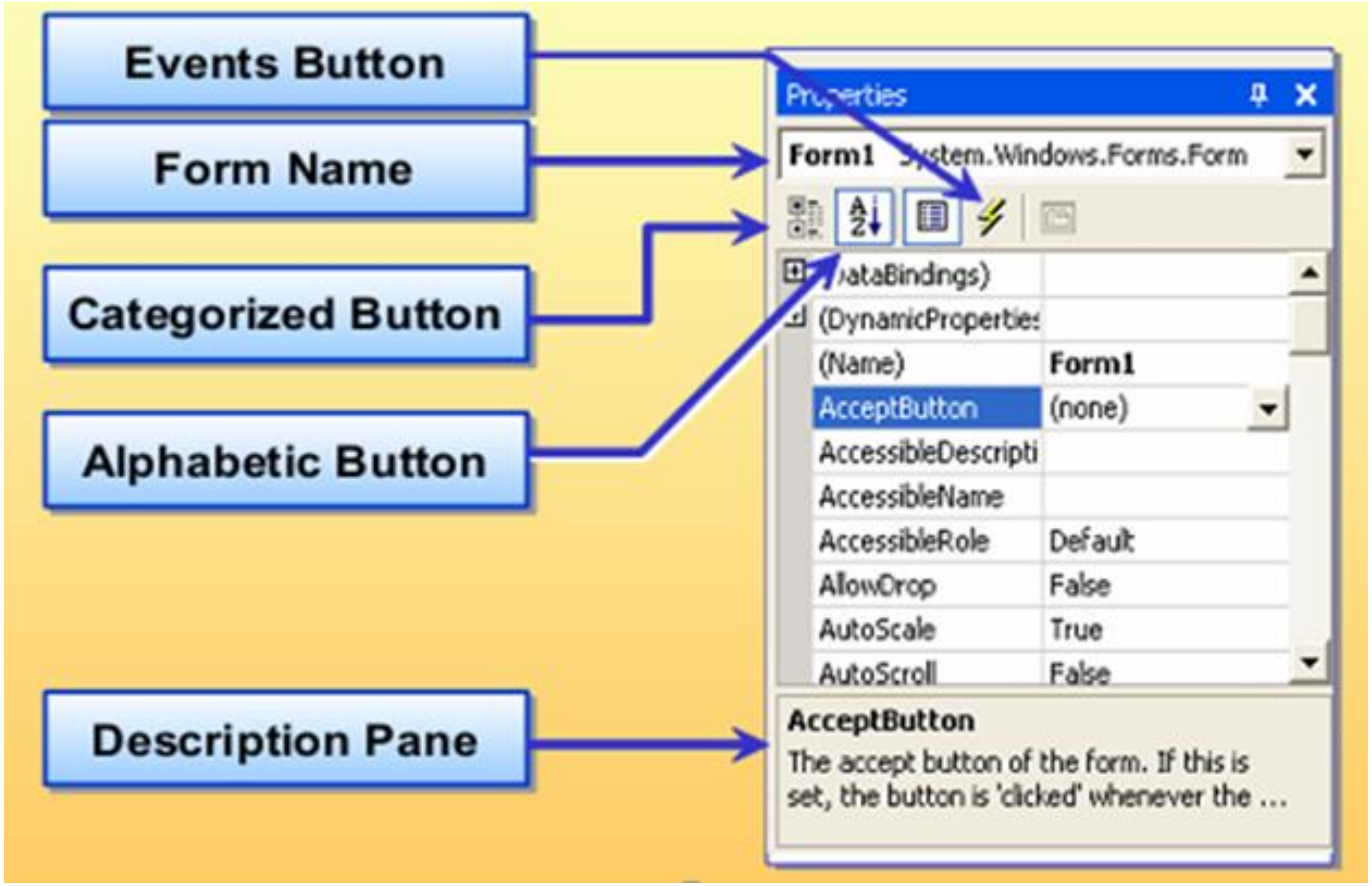
Windows Forms …

- In a Windows-based application, the form is the primary element for user interaction.
- By combining controls and your own actions, you can request information from the user and respond to it.
- In Visual Studio .NET, a form is a window used in your application.
- When you create a new Windows Application project, Visual Studio .NET provides a Designer view that contains a form.
- The default form contains the minimum elements used by most forms:
  - ✓ a title bar,
  - ✓ a control box, and
  - ✓ Minimize, Maximize, and Close buttons.

Windows Forms …

- Most applications require more than one window.
- You must add a form to your project for every window that your application requires.
- **To add additional forms to your project:**
  - ✓ If Solution Explorer is not open, on the **View** menu, click **Solution Explorer** .
  - ✓ In Solution Explorer, **right-click** the project name, point to **Add** , and then click **Windows Form**.
  - ✓ In the Add New Item dialog box, in the Name box, type an appropriate **name** for the form, and then click **Open**.

Forms Properties



(DataBindings)	
(DynamicProperties	
(Name)	Form1
AcceptButton	(none)
AccessibleDescripti	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
AutoScale	True
AutoScroll	False

AcceptButton
The accept button of the form. If this is set, the button is 'clicked' whenever the ...

Form Properties …

- **Name**:-
  - ✓ Sets the name of the form in your project.
  - ✓ This is not the name that is displayed to the user in the caption bar but rather the name that you will use in your code to reference the form.
- **AcceptButton:**
  - ✓ Sets which button is clicked when the user presses the ENTER key.
  - ✓ *Note*: You must have at least one button on your form to use this property.
- **CancelButton:**
  - ✓ Sets which button is clicked when the user presses the ESC key.
  - ✓ *Note*: You must have at least one button on your form to use this property.

Form Properties …

- **ControlBox:**
  - ✓ Determines whether a form displays a control box in the caption bar.
  - ✓ The control box can contain the Minimize button, Maximize button, Help button, and the Close button.
- **FormBorderStyle:**
  - ✓ Controls the appearance of the border for the form.
  - ✓ This will also affect how the caption bar appears and what buttons appear on it.
- **MaximizeBox:**
  - ✓ Determines whether a form has a Maximizebutton in the upper right corner of its caption bar.

Form Properties …

- **MinimizeBox:**
  - ✓ Determines whether a form has a Minimize button in the upper right corner of its caption bar.
- **StartPosition:**
  - ✓ Determines the position of a form on the screen when it first appears.
- **Text;**
  - ✓ Sets the text displayed in the caption bar of the control.
- You can set form properties either by writing **code** or by using the **Properties** window.
- Any property settings that you establish at design time are used as the initial settings each time your application runs.

Form events and methods

- After adding the necessary forms to your project and setting the startup form,
- you must determine which events and methods to use.
- The entire life cycle of a form uses several methods and events.
- When the Show() method is called, the form events and methods are generally triggered in the following order:
  1. Load
  2. GotFocus
  3. Activated
  4. Closing
  5. Closed
  6. Deactivate
  7. LostFocus
  8. Dispose

Form events and methods …

- The **Initialize** event is typically used to prepare an application for use.
- Variables are assigned to initial values, and controls may be moved or resized to accommodate initialization data.
- In .NET, initialization code must be added to the form constructor after the call to **InitializeComponent**() as shown in the following example:

```
public ConstructorName()
{
    // Required for Windows Form Designer support
        InitializeComponent();
  //   Add your initialization code here
}
```

Form Methods …

Modal vs Modeless Forms

- ✓ **Modal**
  - > A form the one where you have to deal with it before you can continue.
- ✓ **Modeless**
  - > You can continue to use other without giving a response

- ➢ **Show**
  - ✓ The Show method includes an implied **Load**;
    - > this means that if the specified form is not already loaded when the **Show** method is called,
      - the application automatically loads the form into memory and then
      - displays it to the user.
  - ✓ The Show method can display forms as **modal** or **modeless**.
    - > example: *formInstanceName.Show();*
  - ✓ You can use the **ShowDialog**() method to show a form as a dialog box.

Form Methods …

- **Load**
  - ✓ The Load event is used to perform actions that must occur before the form displays.
  - ✓ It is also used to assign default values to the form and its controls.
  - ✓ The Load event occurs each time that a form is loaded into memory.
  - ✓ A form’s Load event can run multiple times during an application’s life.
  - ✓ Load fires when a form starts as the result of the
    - > Load statement,
    - > Show statement, or
    - > when a reference is made to an unloaded form’s properties, methods, or controls.

Form Methods …

- **Activated and Deactivate**
  - ✓ When the user moves among two or more forms,
    - you can use the Activated and Deactivate events to define the forms’ behaviors.
  - ✓ The Activated event occurs when the form is activated in code or by the user.
  - ✓ To activate a form at run time by using code, call the Activate method.
  - ✓ You can use this event for tasks such as updating the contents of the form based on changes made to the form’s data when the form was not activated.
  - ✓ The Activated event fires when the form receives focus from another form in the same project.

Form Methods …

- The Activated event fires only when the form is visible.
  - ✓ For example, a form loaded by using the Load statement isn’t visible unless you use the Show method, or set the form’s Visible property to True.
- The Activated event fires before the GotFocus event.
- Use the following code to set the focus to a form.
  - *formInstanceName.Focus*();
- Deactivate fires when the form loses focus to another form. This event fires after the LostFocus event.
- Both the Activated and Deactivate events fire only when focus is changing within the same application.
- If you switch to a different application and then return to the program, neither event fires.

Form Methods …

- **Closing**
  - ✓ The Closing event is useful when you need to know how the user is closing the form.
  - ✓ The Closing event occurs when the form receives a request to close.
  - ✓ Data validation can occur at this time.
  - ✓ If there is a need to keep the form open (for example, if data validation fails), the closing event can be canceled.
- Note
  - ✓ If you need to add code that executes either when
    - > the form being displayed or
    - > the form is being hidden
      - add the code to the Activated and Deactivated event handlers instead of to the GotFocus and LostFocus event handlers

Form Methods …

- **Closed**
  - ✓ The Closed event occurs when the form is closed and before the Dispose event.
  - ✓ Use the Closed event procedure to verify that the form should be closed or to specify actions that take place when closing the form.
  - ✓ You can also include form-level validation code for closing the form or saving data to a file

Form Methods …

- **Dispose**
  - ✓ The .NET framework does not support the Terminate event.
  - ✓ Termination code must execute inside the Dispose method, before the call to base.Dispose().

```
protected override void Dispose( bool disposing )
{
  // Termination code goes here.
 if( disposing )
 {
    if (components != null)
   {
   components.Dispose();
   }
 }
 base.Dispose( disposing );
}
```

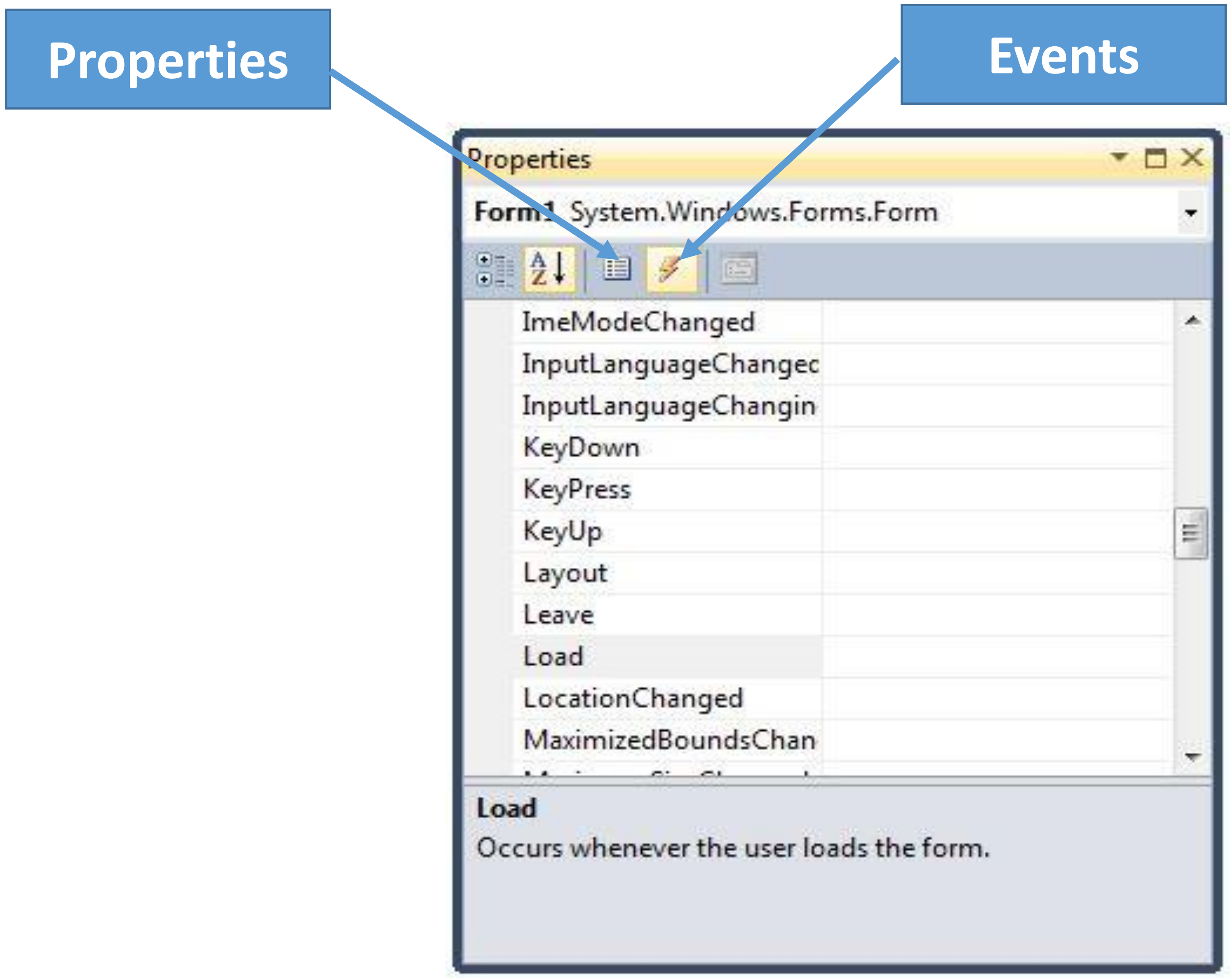
  - ✓ The Dispose method is called automatically for the main form in an application;
  - ✓ you must call it explicitly for any other form.

Form Methods …

- **Hide**
  - ✓ The Hide method removes a form from the screen without removing it from memory.
  - ✓ A hidden form’s controls are not accessible to the user, but they are available to the running application.
  - ✓ When a form is hidden , the user cannot interact with the application until all code in the event procedure that caused the form to be hidden has finished executing.
  - ✓ If the form is not already loaded into memory when the Hide method is called,
    - > the Hide method loads the form but doesn’t display it.

formInstanceName.Hide();

How to Handle Form Events



Form1 System.Windows.Forms.Form

ImeModeChanged
InputLanguageChangec
InputLanguageChangin
KeyDown
KeyPress
KeyUp
Layout
Leave
Load
LocationChanged
MaximizedBoundsChan

Load
Occurs whenever the user loads the form.

Form Events ...

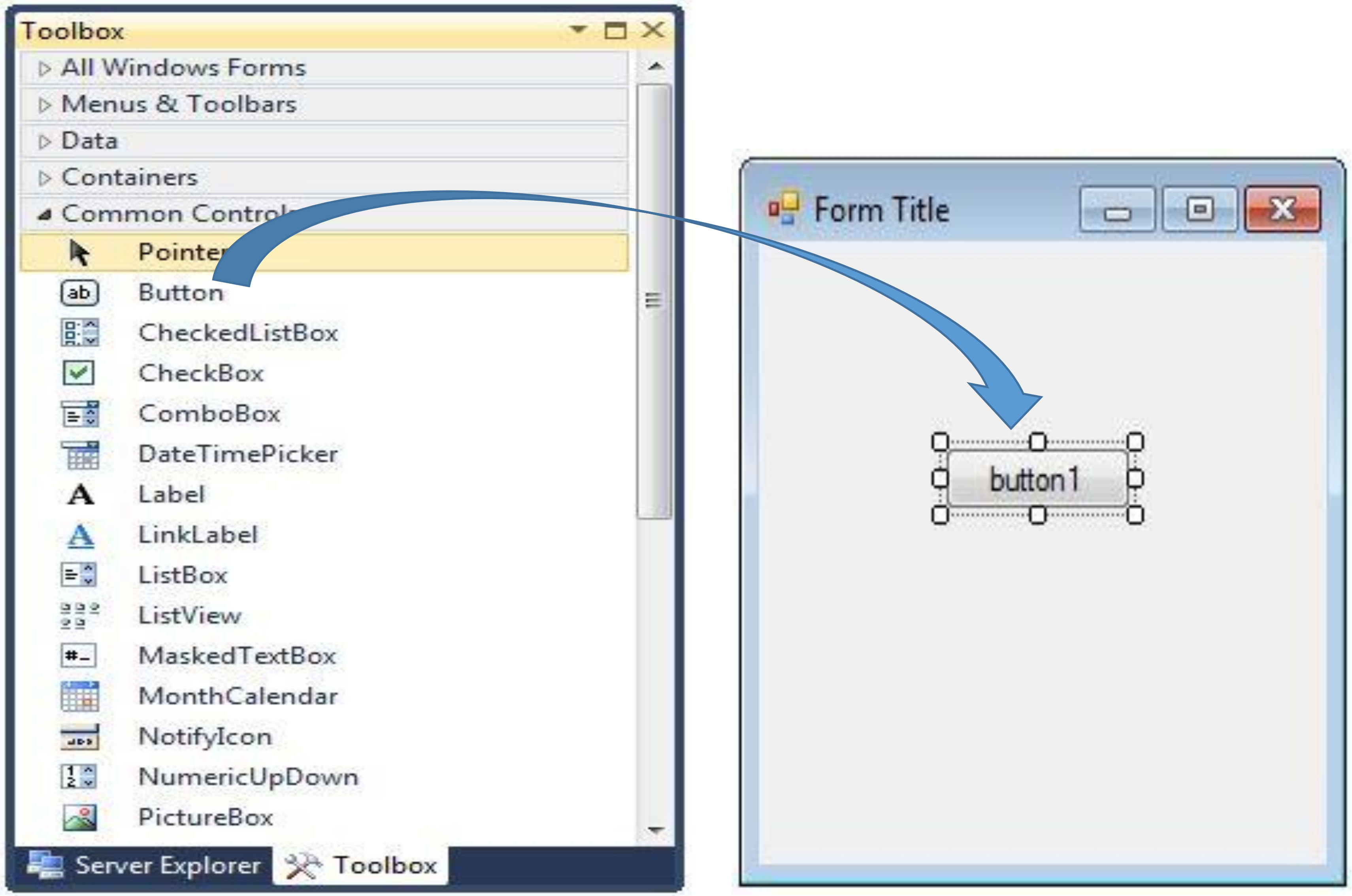
- An ***event handler*** is a segment of code that is called when a corresponding event occurs.
  - ✓ For example,
    - > you can write code in an event handler for the Activated event of a form
      - to perform operations such as updating the data displayed in the controls of the form when the form is activated.
- The .NET Framework uses a standard naming convention for event handlers.
- The *convention* is to combine
  - ✓ the *name of the object* that sends the event,
  - ✓ an *underscore*, and
  - ✓ the *name of the event*.
- For example, the Click event of a form named Form1 would be named ***Form1_Click*** .

Form Events …

1. Open the Properties windows for the form for which you want to add an event handler.
2. Click the Event icon ⚡ in the Properties window to view the events.
3. Double-click the event to add an event handler.
   - ✓ When you create a form by using Windows Forms Designer,
     - > the Designer generates a lot of code that you would have to write if you were creating a form on your own.

- ➢ Note:
  - ✓ Avoid modifying or deleting the Windows Forms Designer generated code.
  - ✓ If you do, can end up with errors in your project

Add Controls to a Form



Controls to a Form …

- Controls are objects that are contained in form objects.
  - ✓ Buttons, TextBoxes, and Labels are examples of controls.
- There are two ways to add controls to a form.
  - ✓ The first way allows you to add several controls quickly and then size and position them individually.
  - ✓ The second way gives you more initial control over the size and position of the control.

Controls to a Form …

To add controls to a form and then size and position them:

1. If the Toolbox is not open, on the View menu, click Toolbox.
2. In the Toolbox, double-click the control that you want to add.
   - ✓ This places an instance of the control at the default size in the upper left corner of the active object.
   - ✓ When adding multiple controls in this manner, they are placed on top of each other.
3. After the controls are added, you can reposition and resize them:
   - ✓ To reposition the control, click the control to select it, and then drag the control to the correct position.
   - ✓ To resize the control, click the control to select it, drag one of the eight sizing handles until the control is properly sized.

Controls to a Form …

To size and position controls while you add them to a form:

1. If the Toolbox is not open, on the View menu, click Toolbox.
2. In the Toolbox, click the control that you want to add.
3. Move the mouse pointer over the form.
   - ✓ The pointer symbol changes to a crosshair.
4. Position the crosshair where you want the upper left corner of the control.
5. Click and drag the crosshair where you want the lower right corner.
   - ✓ A rectangle that indicates the control’s size and location is drawn on the screen.

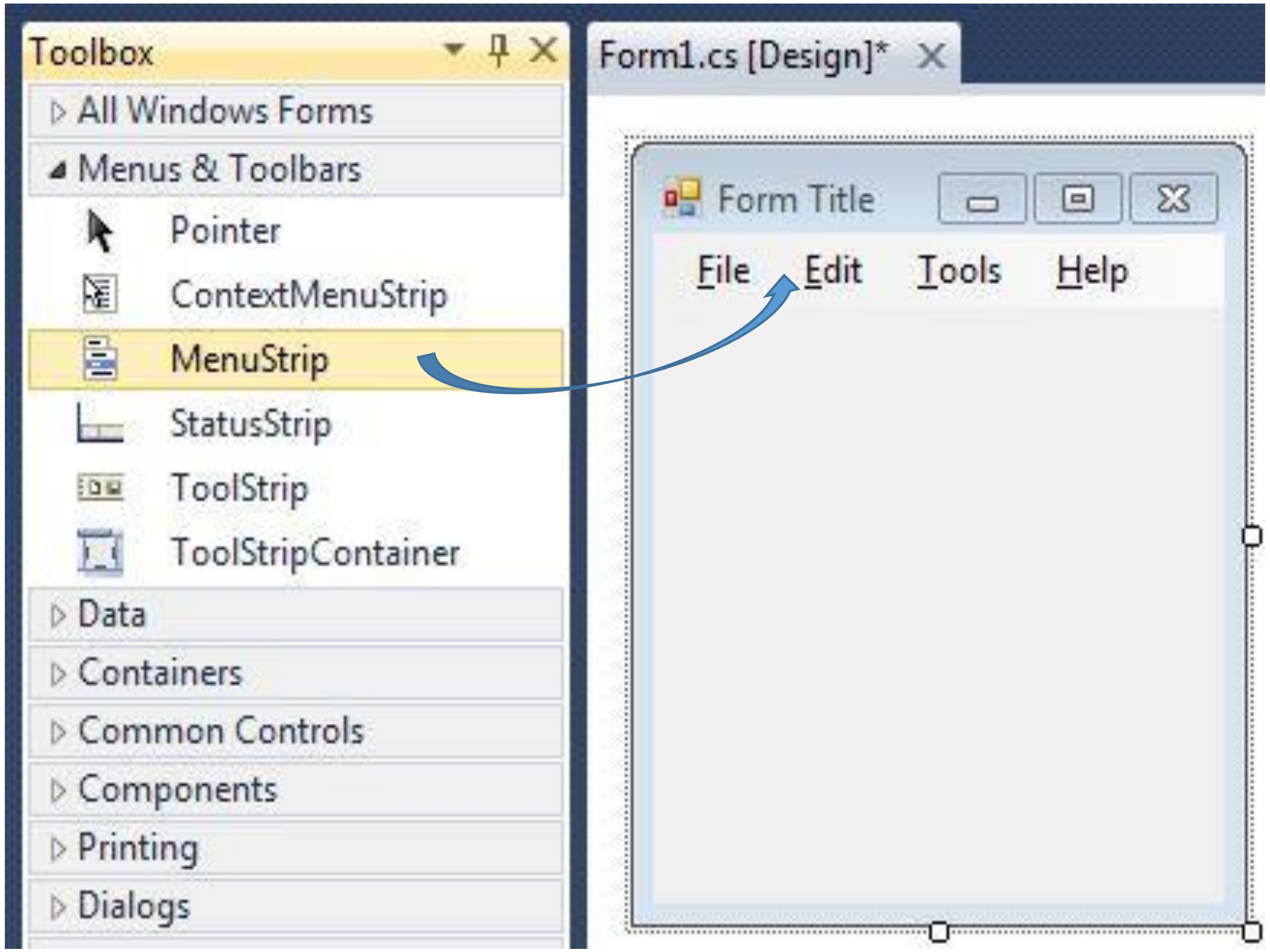
Controls to a Form …

6. When the control is correctly sized, release the mouse button.
   - ✓ The sized control appears in the correct location on the form.
7. You can reposition or resize the control after you have released the mouse button:
   - ✓ To reposition the control, click the control to select it, and then drag the control to the correct position.
   - ✓ To resize the control, click the control to select it, and drag one of the eight sizing handles until the control is properly sized.

How to Add Menus to a Form

- Menus provide a structured way for users to access the commands and tools contained in an application.
- Proper planning and design of menus and toolbars
  - ✓ is essential and
  - ✓ ensures proper functionality and
  - ✓ accessibility of your application to users.

Menu to a form …



Menu to a form …

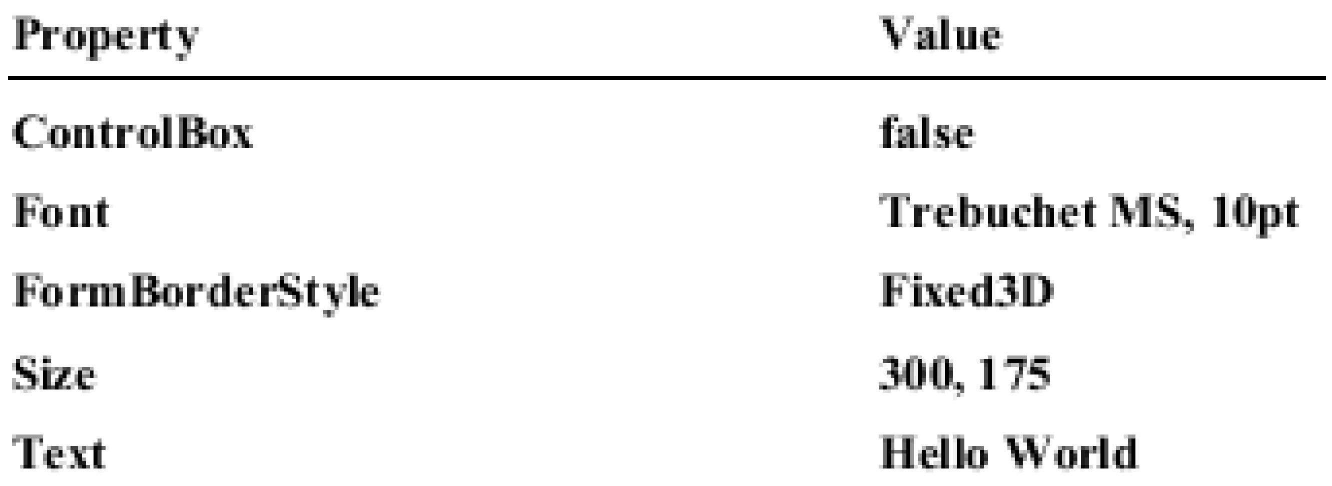
- A menu control properties
  - ✓ **Name**
    - identifies the menu control in code.
  - ✓ **Text**
    - the text that appears on the menu bar at run time.
  - ✓ **Items**
    - collection of items to display on the Toolstrips
  - ✓ To add menus to a form:
    1. If the Toolbox is not open, on the View menu, click Toolbox.
    2. In the Toolbox, double-click the MenuStrip control.
    3. In the Text box, on the newly created menu, type the text for the first menu’s title. This title will appear on the menu bar.
    4. In the Name box, in the Properties window, type the name that you will use to refer to the menu control in code.

Customize the Controls Toolbox

- How to Customize the Controls Toolbox
  - ✓ Right-click the Toolbox
  - ✓ Click-Choose Items
  - ✓ Click the .NET Framework Components tab or the COM Components tab, and
  - ✓ select the required controls.

Creating a Form and Adding Controls

- Set the properties of a form
  1. Create a new project with name- CreatingForms
  2. Open the CreatingForms.cs file in Design view.
  3. If the Properties window is not visible, on the View menu, click Properties Window.
  4. Set the remaining form properties to the indicated values.



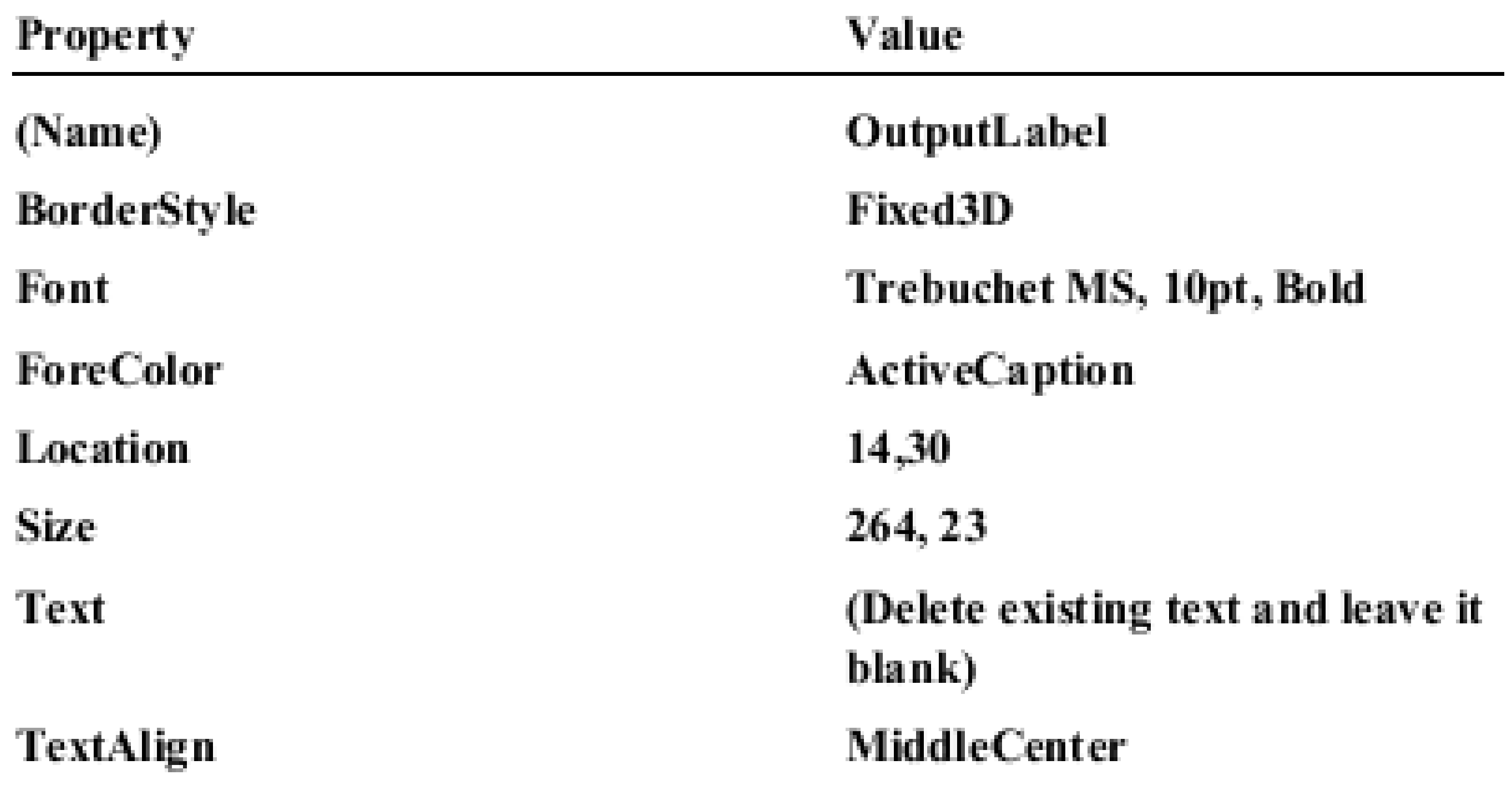
Property	Value
ControlBox	false
Font	Trebuchet MS, 10pt
FormBorderStyle	Fixed3D
Size	300, 175
Text	Hello World

Controls to a form

- Add controls to the form
  1. If the Toolbox is not visible, on the View menu, click Toolbox.
  2. In the Toolbox, double-click the **Label** control to add it to the form.
  3. Double-click the **Button** control to add it to the form.
  4. Double-click the **Button** control again to add a second button to the form.
  5. Position the Label control near the top center of the form.
  6. Position the buttons next to each other near the bottom of the form—with button**1** on the left and button**2** on the right.

Controls Properties

- Set the properties of the controls
  1. Click the Label1 control.
  2. Set the following properties for the Label1 control to the values provided.



Property	Value
(Name)	OutputLabel
BorderStyle	Fixed3D
Font	Trebuchet MS, 10pt, Bold
ForeColor	ActiveCaption
Location	14,30
Size	264, 23
Text	(Delete existing text and leave it blank)
TextAlign	MiddleCenter

Controls Properties …

3. Click Button1.
4. Set the following properties for the Button1 control to the values provided in the following table.

Property	Value
(Name)	HelloButton
Location	57, 87
Size	75,25
Text	&Say Hello

Controls Properties

5. Click Button2.
6. Set the following properties for the Button2 control to the values provided in the following table.

Property	Value
(Name)	ExitButton
Location	161, 87
Size	75,25
Text	E&xit

Controls Properties

7. Double-click the Say Hello button to create the Click event handler.
8. In the Click event handler for HelloButton, add the following line of code:

```
OutputLabel.Text = "Hello, World!";
```

9. Switch back to Design view of the form.
10. Double click the Exit button to create the Click event handler.
11. In the Click event handler for ExitButton , add the following line of code:

```
this.Close();
```

12. Switch back to Design view of the form.
13. Set the AcceptButton property to HelloButton and the CancelButton property to ExitButton .

Build App

- Build and run the application
  1. To build the application, click the Build menu, and then click Build Solution .
  2. Run the application by pressing F5.

Creating an Inherited Form

- Creating new Windows Forms by inheriting from base forms is an efficient way to
  - ✓ duplicate your best efforts without going through the process of entirely recreating a form every time you require it.
- The process of inheriting a form from an existing one is called ***Visual Inheritance***.
- Access Modifiers
  - Private:- Read-only to a child form, all of its property values in the property browser are disabled
  - Protected:- Accessible within the class and from any class that inherits from the class that declared this member
  - Public:-Most permissive level.
    - Public controls have full accessibility

Inherited Forms …

- There are two ways of implementing Visual Inheritance in a Visual Studio .NET project.
  - ✓ To create an inherited form programmatically:

1. Create a new project in Visual Studio .NET.
2. Add another form, or view the code of Form1, which is created by default.
3. In the class definition, add a reference to the form to inherit from.
   - > The reference should include the namespace that contains the form, followed by a period, and then the name of the base form itself.

   ```
   public class Form2 : Namespace1.Form1
   ```

   - > The form now takes on the characteristics of the inherited form.
   - > It contains the controls that were on the inherited form, the code, and the properties.

Inherited Forms …

- To create an inherited form by using the Inheritance Picker dialog box:
  1. On the Project menu, click Add Windows Form .
  2. In the Categories pane, click Windows Form Items, and in the Templates pane, click Inherited Form .
     - In the Name box, type a name, and then click Open.
     - This will open the Inheritance Picker dialog box.
  3. Click Browse, and locate the compiled executable of the project that contains your form.
     - Click OK.
     - The new form should now be added into the project and be based on your inherited form.
     - It contains the controls on the base form.

Organizing Controls on a Form

- How to Arrange Controls on a Form by Using the Format Menu
- How to Set the Tab Order for Controls
- How to Anchor a Control in Windows Forms
- How to Dock a Control in Windows Forms
- How to resize and position controls

Organizing Controls …

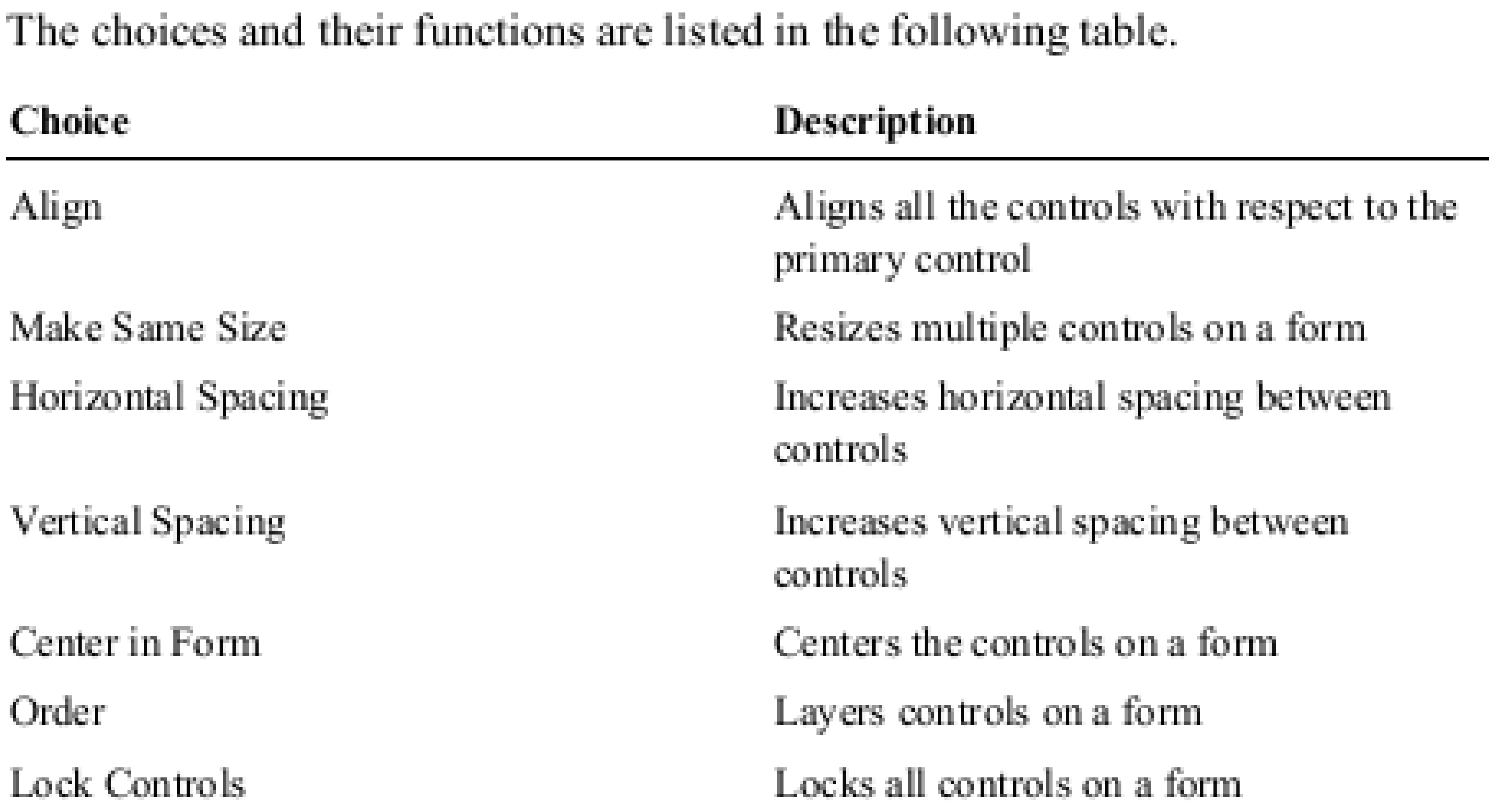
- Often you want to change the position and dimensions of controls at run time.
  - ✓ Currently, you handle the resize event
    - calculate the new position, width, and height, and
    - call some methods like Move or SetWindowPos.
- The Windows Forms library offers two very helpful concepts to simplify these proceedings:
  - anchoring and
  - docking.
- In addition, Visual Studio .NET provides the Format menu,
  - ✓ which allows you to arrange controls on a form.

Organizing Controls …

Using the Format Menu

- You can use the Format menu or the Layout Toolbar
  - ✓ to align, layer, and lock controls on a form.
- The Format menu provides many options for organizing controls.
- When you use the Format menu options for organizing controls,
  - ✓ select controls so that the last control that you select is the primary control to which the others are aligned.
  - ✓ The primary control has dark sizing handles whereas other controls have light sizing handles around them.

Organizing Controls …

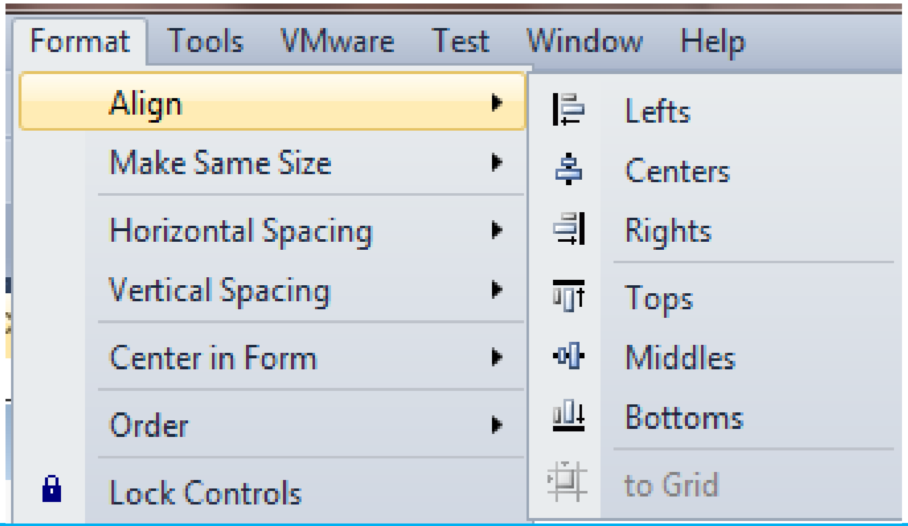


The choices and their functions are listed in the following table.

Choice	Description
Align	Aligns all the controls with respect to the primary control
Make Same Size	Resizes multiple controls on a form
Horizontal Spacing	Increases horizontal spacing between controls
Vertical Spacing	Increases vertical spacing between controls
Center in Form	Centers the controls on a form
Order	Layers controls on a form
Lock Controls	Locks all controls on a form

Organizing Controls …

1. In the Windows Forms Designer,
   - ✓ open the form that contains the controls that you want to position.
2. Select the controls that you want to align
   - ✓ so that the last control that you select is the primary control to which the others are aligned.
   - ✓ On the Format menu, point to Align, and then click any of the seven choices available.



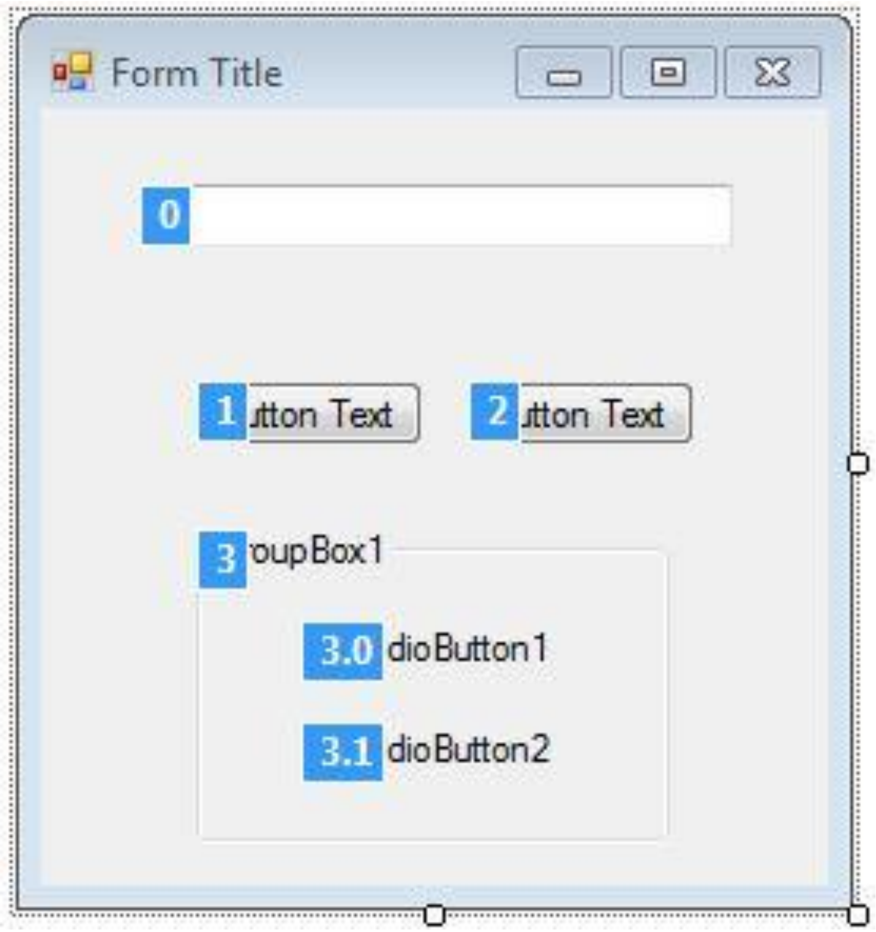
Organizing Controls …

- When creating complex user interfaces, you may want to layer controls on a form.
  - ✓ To layer controls on a form:
    1. Select a control.
    2. On the Format menu, point to Order, and then click Bring To Front or Send To Back .
  - ✓ You can lock all controls on a form.
    - This prevents any accidental moving or resizing of controls if you are setting other properties for controls.
  - ✓ To lock all controls on a form,
    - on the Format menu, click Lock Controls.

Organizing Controls …

- **How to Set the Tab Order for Controls**
  - ✓ On the View menu, select Tab Order
  - ✓ Click a control to change its tab order

  Or
  - ✓ Set the TabIndex property
  - ✓ Set the TabStop property to True



Organizing Controls …

- The tab order is the order in which a user moves focus from one control to another by pressing the **TAB** key.
- Each form has its own tab order.
- By default, the tab order is the same as the order in which you created the controls.
- Tab order numbering begins with zero.
- You can set tab order in the Properties window by using the TabIndex property.
- The TabIndex property of a control determines where it is positioned in the tab order.
- By default, the first control drawn has a TabIndex value of 0 , the second has a TabIndex of 1 , and so on.

Organizing Controls …

- To set the tab order by using the View menu:
  1. On the View menu, click Tab Order.
  2. Click the controls sequentially to establish the tab order that you want.
  3. When you are finished, on the View menu, click Tab Order.
- To set the tab order by using the TabIndex property:
  1. Select the control.
  2. Set the TabIndex property to the required value.
  3. Set the TabStop property to True.
- By turning off the **TabStop** property,
  - ✓ you enable a control be passed over in the tab order of the form.
- A control in which the TabStop property has been set to False
  - ✓ still maintains its position in the tab order,
  - ✓ even though the control is skipped when you cycle through the controls with the TAB key.

Organizing Controls …

- **Anchoring**
  - ✓ Ensures that the edges of the control remain in the same position with respect to the parent container
- **To anchor a control to the form**
  - ✓ Set its **Anchor** property
  - ✓ Default value: **Top**, **Left**
  - ✓ Other Styles: **Bottom**, **Right**

Organizing Controls …

- If you are designing a form that the user can resize at run time,
  - ✓ the controls on the form should resize and reposition properly.
- When a control is anchored to a form (or another container) and the form is resized,
  - ✓ the control maintains the distance between the control and the anchor positions (which is the initial position).
- You use the **Anchor** property editor to anchor.

Organizing Controls …

- To anchor a control on a form:
  1. Select the control that you want to anchor.
  2. In the **Properties** window,
     - click the **Anchor** property, and then
     - click the **Anchor** arrow.
     - The **Anchor** property editor is displayed; it contains a top bar, left bar, right bar, and bottom bar.
  3. To set an anchor, click the top, left, right, or bottom bar in the **Anchor** property editor.
- Controls are anchored to the top and left by default.
- To clear a side of the control that has been anchored, click the bar on that side.

Organizing Controls …

- **Docking**
  - ✓ Enables you to glue the edges of a control to the edges of its parent control
  - ✓ **To dock a control**
  - ✓ Set the Dock property

Organizing Controls …

- You can dock controls to the edges of your form.
- For example,
  - ✓ Windows Explorer docks
    - the **TreeView** control to the left side of the window and
    - the **ListView** control to the right side of the window.
- Use the **Dock** property for all visible Windows Forms controls to define the docking mode.
- If you use the **Dock** property, a control is coupled with two edges of the container.
- Then, the control is horizontally or vertically resized when the container is resized.

Organizing Controls …

- In real life, you do not indicate the edges, you indicate a border.
- If you dock a control to the left border,
  - ✓ it is connected with the top left and bottom left edge.
- The value of the **Dock** property is one of the **DockStyle** values.
- A special case is **DockStyle.Fill**.
  - ✓ This value docks the control to all edges.
  - ✓ The control fills in the complete client area of the container.

Organizing Controls …

- To dock a control on a form:
  1. Select the control that you want to dock.
  2. In the Properties window, click the arrow to the right of the **Dock** property.
     - The **Dock** property editor is displayed; it contains a series of buttons that represent the edges and the center of the form.
  3. Click the button that represents the edge of the form where you want to dock the control.
     - To fill the contents of the control’s form or container control, click the **Fill** (center) button.
     - Click **None** to disable docking.

  ✓ The control is automatically resized to fit the boundaries of the docked edge.

MDI Applications

- When creating Windows-based applications,
  - ✓ you can use different styles for the user interface.
- Your application can have
  - ✓ a single-document interface (SDI) or
  - ✓ a multiple-document interface (MDI), or
  - ✓ you can create an explorer-style interface.
- For more information about different types of application interfaces, see the .NET Framework software development kit (SDK).

SDI Vs MDI

SDI

- Only one document is visible at a time.
- Must close one document before opening another.
- **Example**:
  - ✓ Microsoft WordPad

MDI

- Several documents are visible at the same time.
- Each document is displayed in its own window.
- **Example**:
  - ✓ Microsoft Excel

Creating MDI Applications

- To create a parent form
  - ✓ Create a new project
  - ✓ Set the IsMdiContainer property to True
  - ✓ Add a menu item to invoke the child form
- To create a child form
  - ✓ Add a new form to the project
- To call a child form from a parent form

```
protected void MenuItem1_onClick(object sender, EventArgs e)
{
    Form2 newChildForm = new Form2();
    //set parent Form
    newChildForm.MdiParen = this;
    //display the child form
    newChildForm.Show();
}
```

Creating MDI Application …

- There are three main steps involved in creating an MDI application:
  1. creating a parent form,
  2. creating a child form, and
  3. calling the child form from a parent form.
- To create the parent form at design time:
  - ✓ The Parent form in an MDI application is the form that contains the MDI child windows.
  - ✓ Child windows are used for interacting with users in an MDI application.

1. Create a new project.
2. Select the default Form, In the Properties window, set the IsMdiContainer of the Form property to True.
   - ✓ This designates the form as an MDI container for child windows.

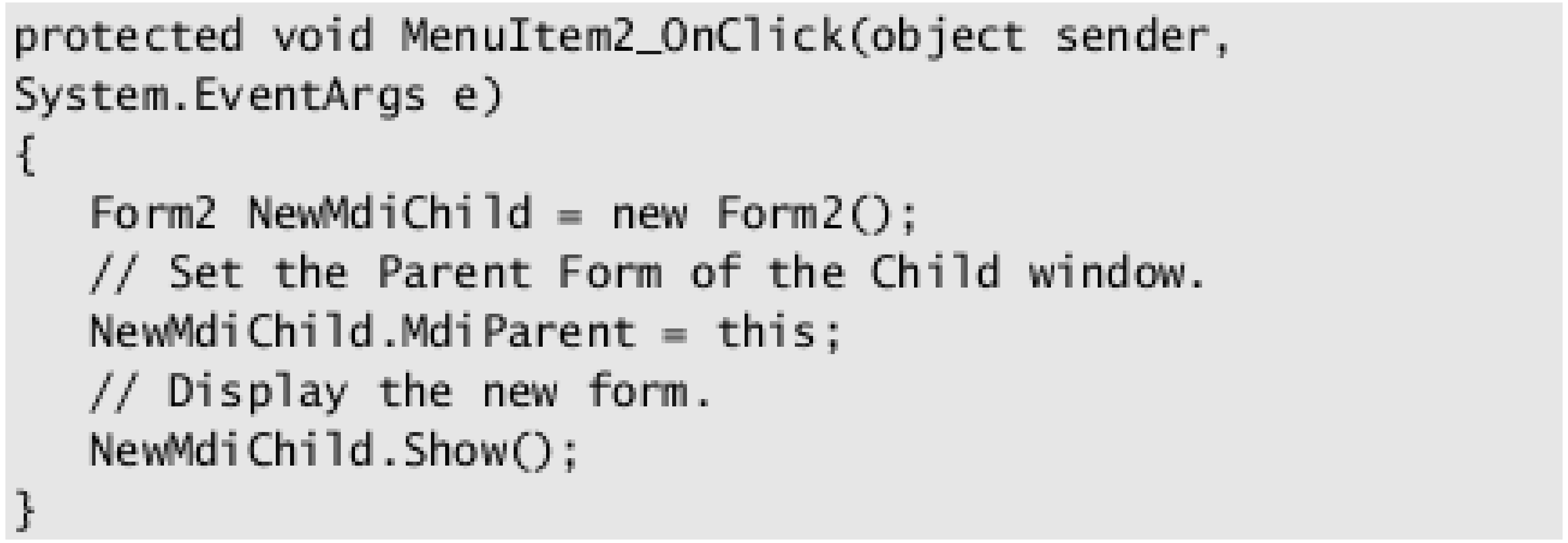
Creating MDI Application ...

3. From the Toolbox, drag a MainMenu component to the form.
   - You need a menu to invoke the child forms from the parent form.
4. Set the MdiList property of the Windows menu item to True.

- MDI child forms are critical for MDI applications because users interact with the application through child forms.
- To create the child form at design time:
  - ✓ In the same project that contains the parent form, create a new form.
- You can also create an MDI form by
  - ✓ Click Project Menu/Right click on Project
    - Add => Add Windows Form
  - ✓ Select MDI Parent Form
  - ✓ Set the MDI Form as a startup Form

Creating MDI Application …

- To call the child form from the parent form:
  1. Create a Click event handler for the New menu item on the parent form.
  2. Insert code similar to the following code to create a new MDI child form when the user clicks the New menu item.



```
protected void MenuItem2_OnClick(object sender,
System.EventArgs e)
{
   Form2 NewMdiChild = new Form2();
   // Set the Parent Form of the Child window.
   NewMdiChild.MdiParent = this;
   // Display the new form.
   NewMdiChild.Show();
}
```

Creating MDI Application …

- How Parent and Child Forms Interact
  - ✓ To list the available child windows that are owned by the parent
    - Create a menu item (Windows) and set its MdiList property to True
  - ✓ To determine the active MDI child
    - Use the ActiveMdiChild property
  - ✓ To arrange child windows on the parent form
    - Call the LayoutMdimethod

Creating MDI Application ...

- In an MDI application,
  - ✓ a parent form has several child forms, and
  - ✓ each of the child forms interacts with the parent form.
  - ✓ Visual Studio .NET includes several properties that allow parent and child forms in an MDI application to interact.
- An easy way to keep track of the different MDI child windows an application has open is to use a Window list.
- The functionality to keep track of all
  - ✓ the open MDI child forms as well as which child form has focus is part of Visual Studio .NET and is set with the MdiList property of a menu item.

Creating MDI Application ...

- **To list the child windows of a parent form by using the Window list:**

1. Add a MainMenu component to the parent form.
2. Add the following top-level menu items to the MainMenu component by using the Menu Designer.

MenuItem	Text
menuItem1	&File
menuItem2	&Windows

3. Set the MdiList property of the Windows menu item to True.

Creating MDI Application ...

- When you are completing certain procedures in an application,
  - ✓ it is important to determine the active form.
- Because an MDI application can have many instances of the same child form,
  - ✓ the procedure must know which form to use.
- To specify the correct form,
  - ✓ use the ActiveMdiChild property,
    - which returns the child form that has the focus or that was most recently active.
- Use the following code to determine the active child form.

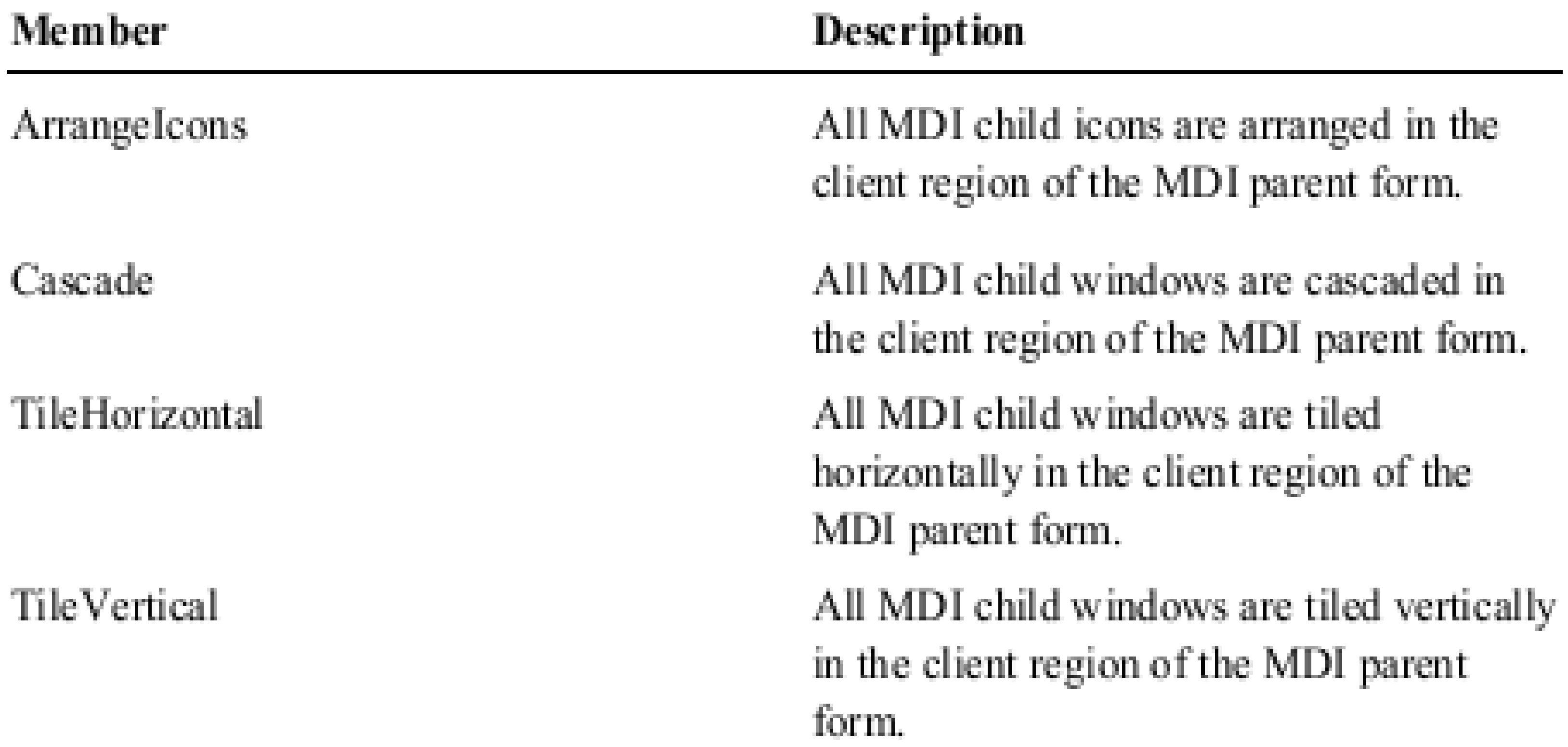
```
Form activeChild = this.ActiveMdiChild;
```

Creating MDI Application …

- Arranging child windows on the parent form
  - ✓ To arrange child windows on a parent form,
    - > you can use the LayoutMdi method with the MdiLayout enumeration to rearrange the child forms in an MDI parent form.
  - ✓ There are four different MdiLayout enumeration values that can be used by the LayoutMdi method.
  - ✓ These values help you display the form as
    - > cascading,
    - > horizontally or
    - > vertically tiled, or
    - > as child form icons arranged along the lower portion of the MDI form.
  - ✓ To arrange child forms, in an event,
    - > use the LayoutMdi method to set the MdiLayout enumeration for the MDI parent form.

Creating MDI Application ...

- You can use the following members of the MdiLayout enumeration when calling the LayoutMdi method of the Form class.



Member	Description
ArrangeIcons	All MDI child icons are arranged in the client region of the MDI parent form.
Cascade	All MDI child windows are cascaded in the client region of the MDI parent form.
TileHorizontal	All MDI child windows are tiled horizontally in the client region of the MDI parent form.
TileVertical	All MDI child windows are tiled vertically in the client region of the MDI parent form.

Creating MDI Application ...

- The following example uses the Cascade setting of the MdiLayout enumeration for the child windows of the MDI parent form (Form1).

```
protected void CascadeWindows_Click(object sender, System.EventArgs e)
{
    Form1.LayoutMdi(MdiLayout.Cascade);
}
```

More information on

- John Sharp. Microsoft Visual C# 2013 Step by Step, 2015 Microsoft Press USA
- Joel Murach, Anne Boehm. Murach C# 2012, Mike Murach & Associates Inc USA, 2013
- Microsoft Corporation. Microsoft Visual Studio 2012 Product Guide, 2012

QUESTIONS

?