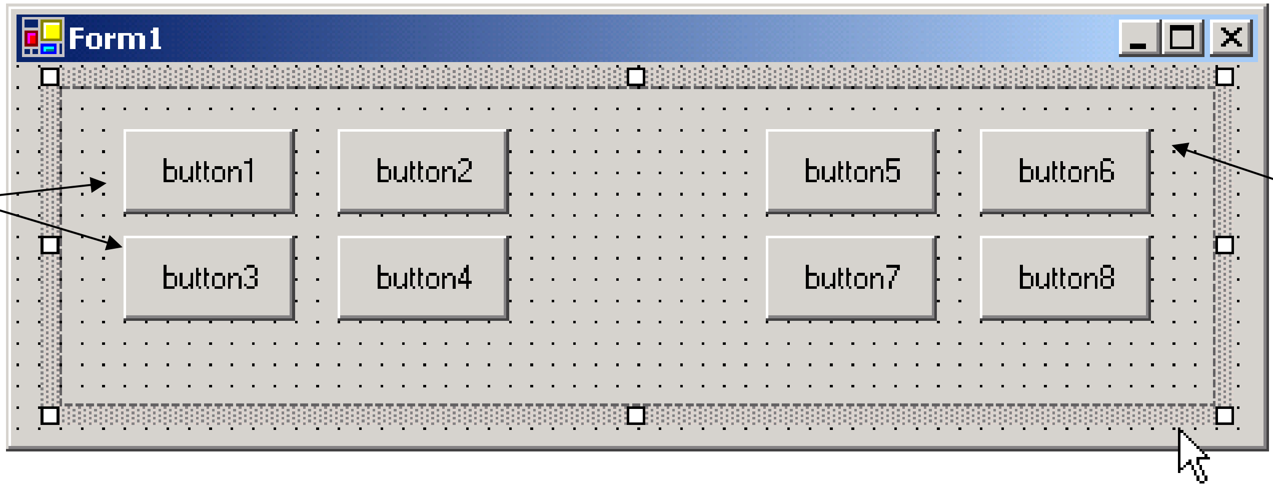
Chapter 12 - Graphical User Interface Concepts: Part 2

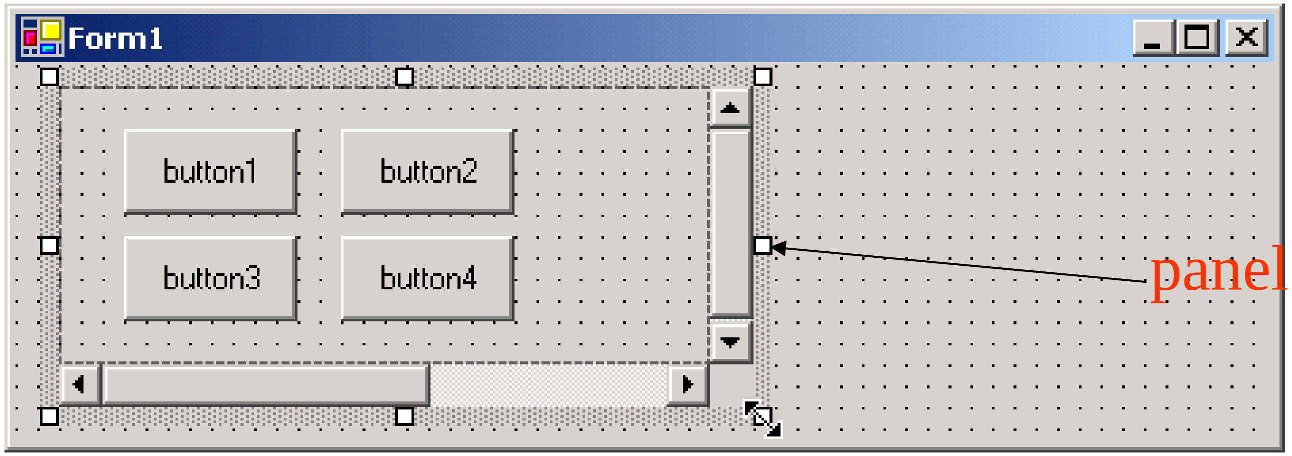
12.6 GroupBoxes and Panels

- Arrange components on a GUI
  - **Panels** can have scrollbar
    - View additional controls inside the Panel

Controls inside panel



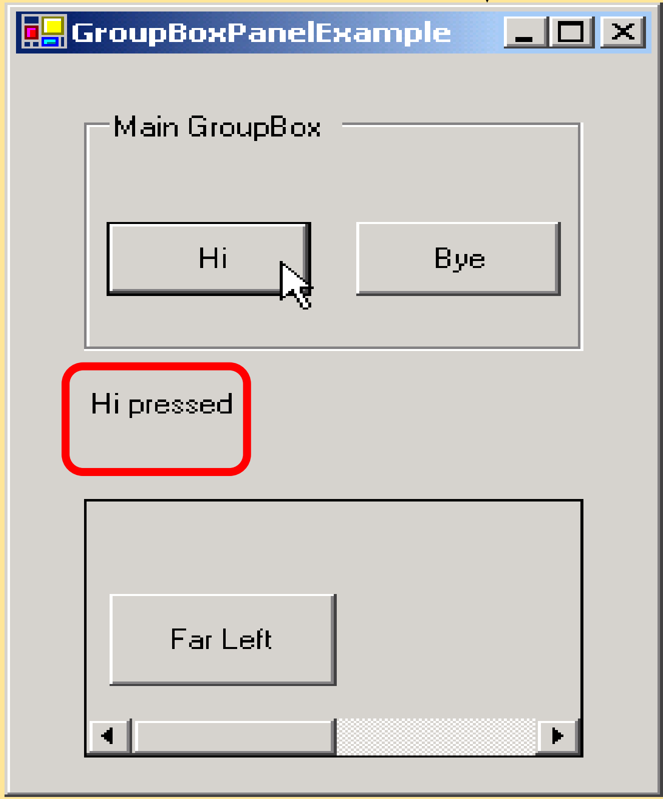
panel



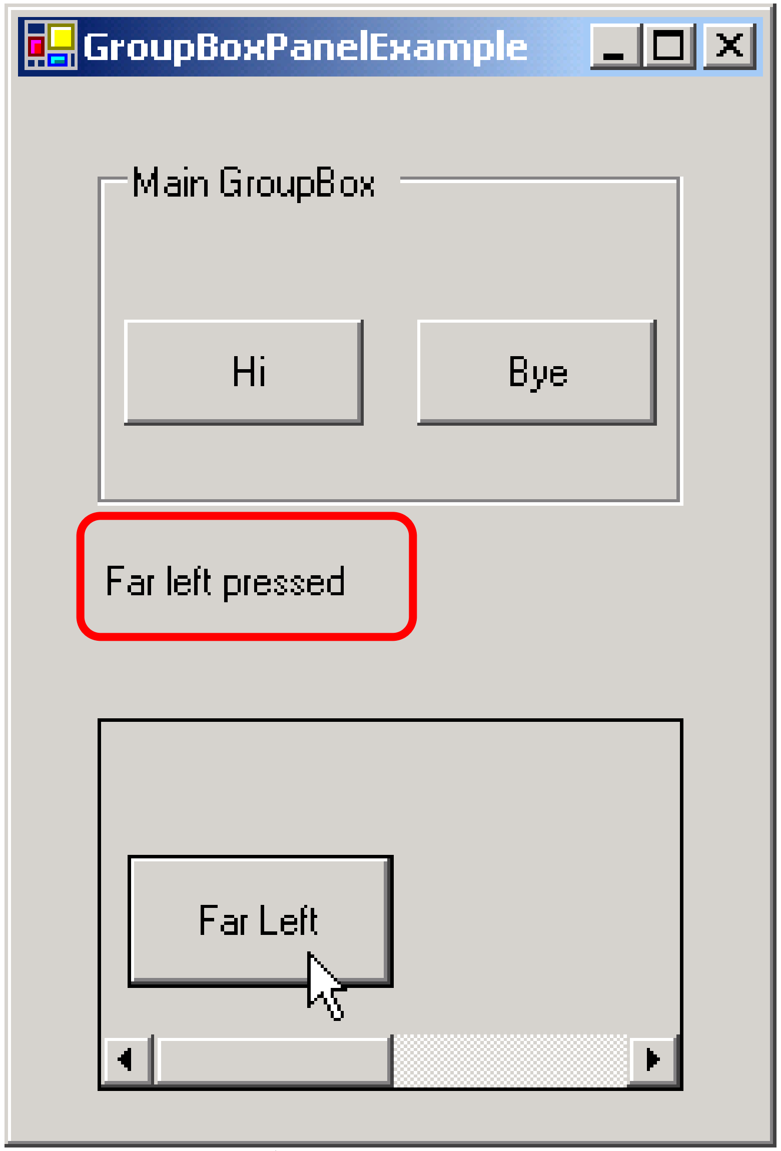
panel scrollbars

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
public class GroupBoxPanelExample:System.Windows.Forms.Form {
    private System.Windows.Forms.Button hiButton;
    private System.Windows.Forms.Button byeButton;
    private System.Windows.Forms.Button leftButton;
    private System.Windows.Forms.Button rightButton;
    private System.Windows.Forms.GroupBox mainGroupBox;
    private System.Windows.Forms.Label messageLabel;
    private System.Windows.Forms.Panel mainPanel;
    private System.ComponentModel.Container components = null;
    [STAThread]
    static void Main() {
        Application.Run( new GroupBoxPanelExample() );
    }

    private void hiButton_Click(
      object sender, System.EventArgs e ) {
        messageLabel.Text= "Hi pressed";
    }
    private void byeButton_Click(
    object sender, System.EventArgs e ) {
        messageLabel.Text = "Bye pressed";
    }
    private void leftButton_Click(
     object sender, System.EventArgs e ) {
        messageLabel.Text = "Far left pressed";
    }
    private void rightButton_Click(
    object sender,System.EventArgs e){
        messageLabel.Text =
         "Far right pressed";
    }
}
```

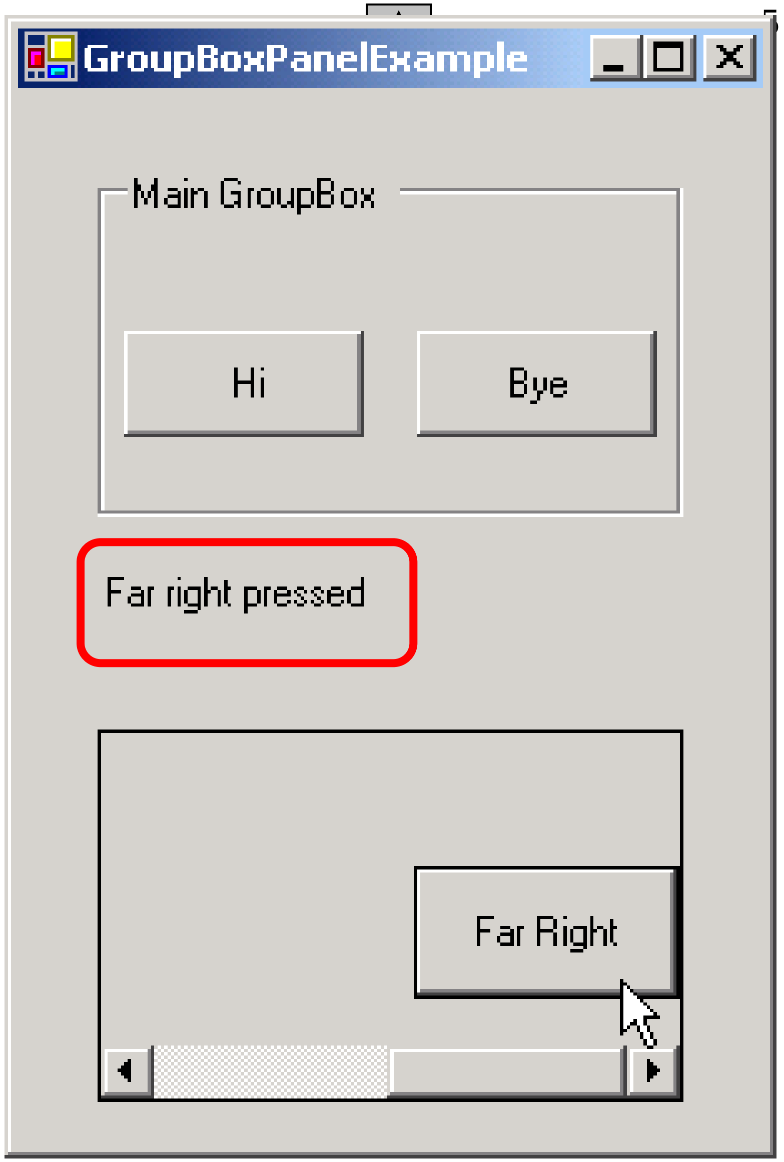
hiButton_Click



Hi pressed



leftButton_Click



rightButton_Click

12.7 Checkboxes and RadioButtons

- State buttons
  - On/off or true/false state
  - Two buttons derived from class ButtonBase
    - CheckBox: usually for multiple choice
    - RadioButton: usually for single choice
- A font is a class with three attributes
  - i.e., name, size, style
    - A style can have bold, italic, strikeout, regular
- FontStyle.Bold and FontStyle.Italic are constant (= 1) defined beforehand
- ^ is an XOR operation, i.e.,

  1 ^ 1 = 0;

  0 ^ 1 = 1

Java's `Font`-related methods and constants

Method or constant	Description
`Font` *constants, constructors and methods for drawing polygons*	
`public final static int PLAIN`	A constant representing a plain font style.
`public final static int BOLD`	A constant representing a bold font style.
`public final static int ITALIC`	A constant representing an italic font style.
`public Font( String name, int style, int size )`	Creates a `Font` object with the specified font, style and size.
`public int getStyle()`	Returns an integer value indicating the current font style.
`public int getSize()`	Returns an integer value indicating the current font size.
`public String getName()`	Returns the current font name as a string.
`public String getFamily()`	Returns the font's family name as a string.
`public boolean isPlain()`	Tests a font for a plain font style. Returns `true` if the font is plain.
`public boolean isBold()`	Tests a font for a bold font style. Returns `true` if the font is bold.
`public boolean isItalic()`	Tests a font for an italic font style. Returns `true` if the font is italic.

Style is integer

font **is class**

Java's Font Control

- Class `Font`
  - Contains methods and constants for font control
  - Font constructor takes three arguments
    1. Font name
       - `Monospaced`, `SansSerif`, `Serif`, etc.
    2. Font style
       - `Font.PLAIN`, `Font.ITALIC` and `Font.BOLD`
    3. Font size
       - Measured in points (1/72 of inch)

Class Font

final static int PLAIN

final static int BOLD

final static int ITALIAN

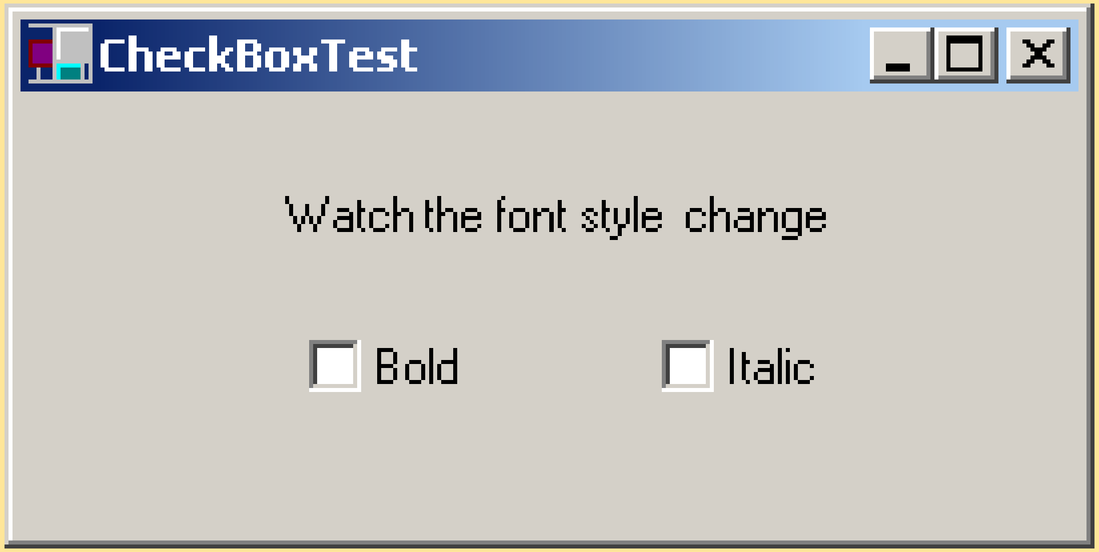
string fontName
int style
int size

Font( )

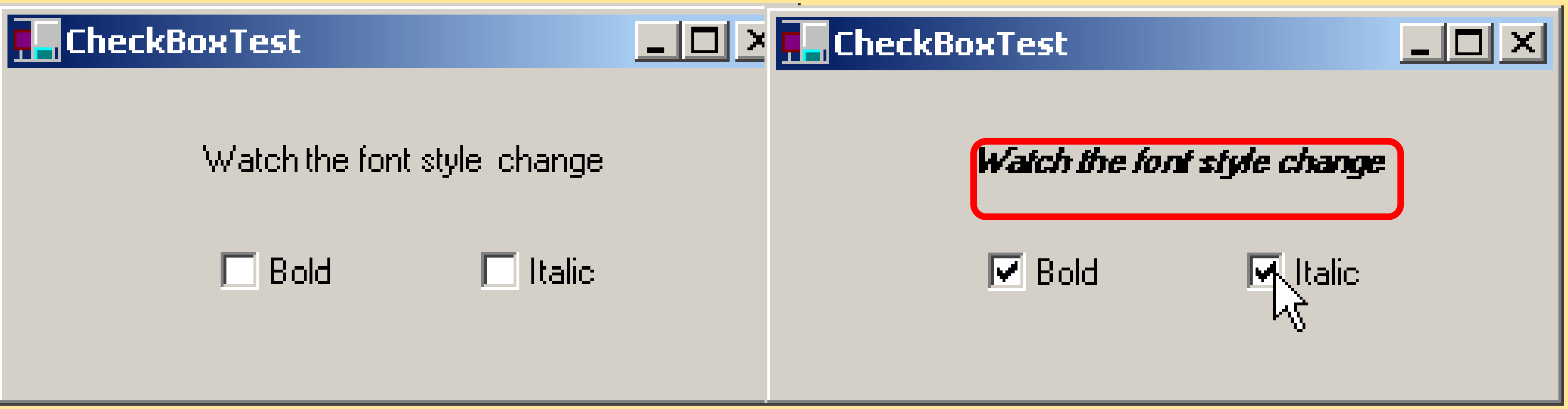
GetStyle ( )
GetSize ( )

…..

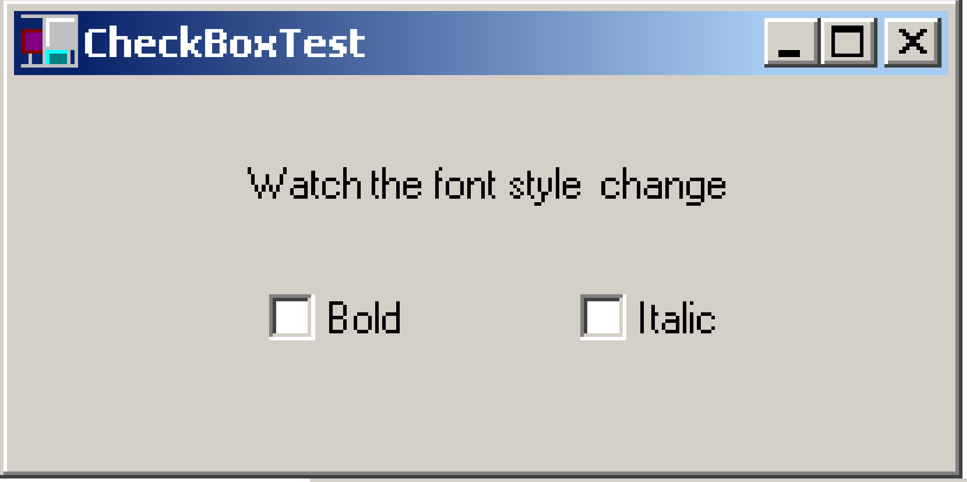
```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
 public class CheckBoxTest : System.Windows.Forms.Form {
    private System.Windows.Forms.CheckBox boldCheckBox;
    private System.Windows.Forms.CheckBox italicCheckBox;
    private System.Windows.Forms.Label outputLabel;
    private System.ComponentModel.Container components = null;
    [STAThread]
    static void Main() {
       Application.Run( new CheckBoxTest() );
    }

```

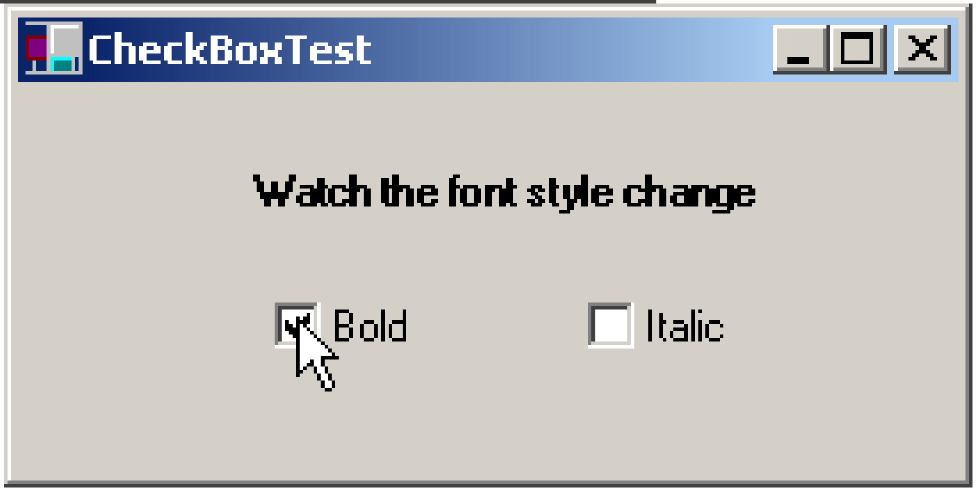


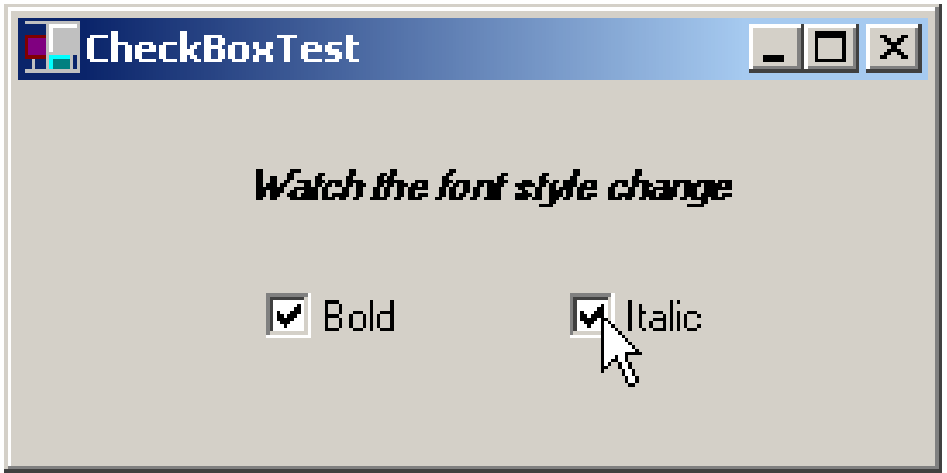
```
   private void boldCheckBox_CheckedChanged(
      object sender, System.EventArgs e ) {
      outputLabel.Font =
         new Font( outputLabel.Font.Name, outputLabel.Font.Size,
            outputLabel.Font.Style ^ FontStyle.Bold );
   }
   private void italicCheckBox_CheckedChanged(
      object sender, System.EventArgs e ) {
      outputLabel.Font =
         new Font( outputLabel.Font.Name, outputLabel.Font.Size,
            outputLabel.Font.Style ^ FontStyle.Italic );
   }
}
```



CheckBoxTest.cs
Program Output

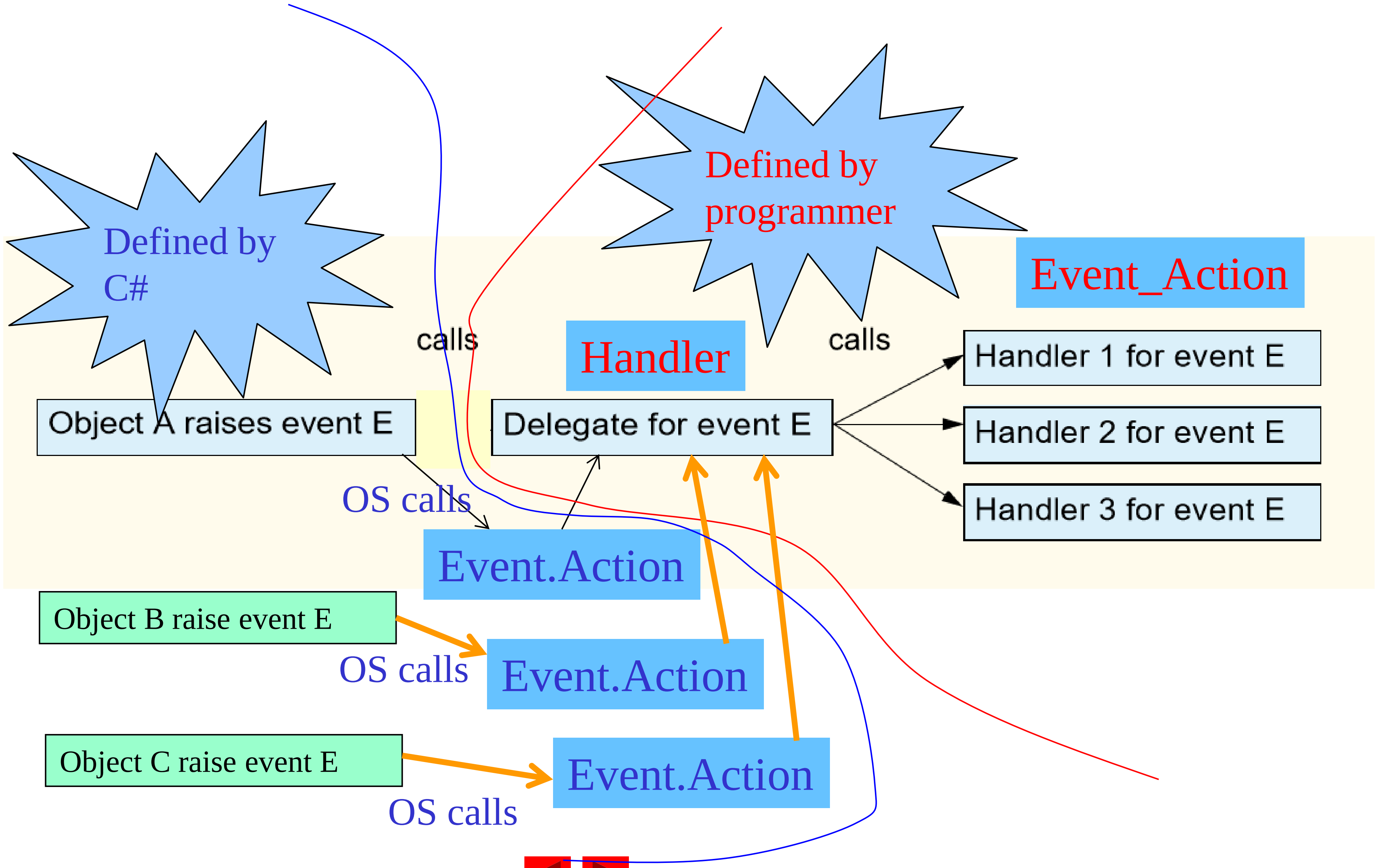






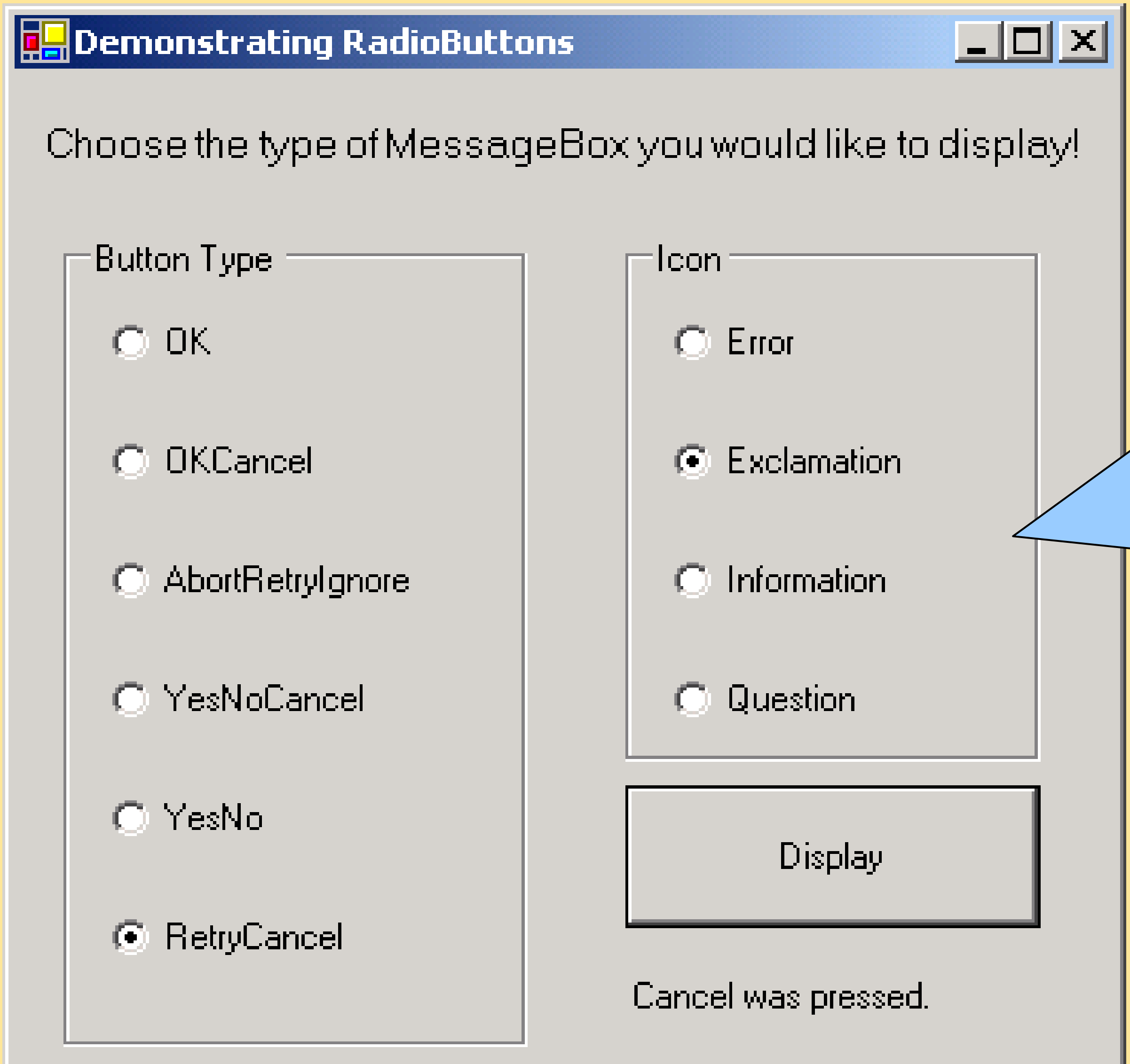
`RadioButton` **properties and events**	**Description / Delegate and Event Arguments**
Common Properties	
`Checked`	Whether the **`RadioButton`** is checked.
`Text`	Text displayed to the right of the **`RadioButton`** (called the label).
Common Events	*(Delegate* **`EventHandler`***, event arguments* **`EventArgs`***)*
`Click`	Raised when user clicks the control.
`CheckedChanged`	Raised every time the **`RadioButton`** is checked or unchecked. Default event when this control is double clicked in the designer.

C# Event's Framework



```csharp
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
 public class RadioButtonsTest : System.Windows.Forms.Form {
    private System.Windows.Forms.Label promptLabel;
    private System.Windows.Forms.Label displayLabel;
    private System.Windows.Forms.Button displayButton;
    private System.Windows.Forms.RadioButton questionButton;
    private System.Windows.Forms.RadioButton informationButton;
    private System.Windows.Forms.RadioButton exclamationButton;
    private System.Windows.Forms.RadioButton errorButton;
    private System.Windows.Forms.RadioButton retryCancelButton;
    private System.Windows.Forms.RadioButton yesNoButton;
    private System.Windows.Forms.RadioButton yesNoCancelButton;
    private System.Windows.Forms.RadioButton okCancelButton;
    private System.Windows.Forms.RadioButton okButton;
```

```
        private System.Windows.Forms.RadioButton abortRetryIgnoreButton;
        private System.Windows.Forms.GroupBox groupBox2;
        private System.Windows.Forms.GroupBox groupBox1;
        private MessageBoxIcon iconType = MessageBoxIcon.Error;
        private MessageBoxButtons buttonType = MessageBoxButtons.OK;
```



Demonstrating RadioButtons

Choose the type of MessageBox you would like to display!

Button Type

- ○ OK
- ○ OKCancel
- ○ AbortRetryIgnore
- ○ YesNoCancel
- ○ YesNo
- ● RetryCancel

Icon

- ○ Error
- ● Exclamation
- ○ Information
- ○ Question

Cancel was pressed.

Radio buttons are put in a GroupBox by visual setting (setting codes in initializeComponent)

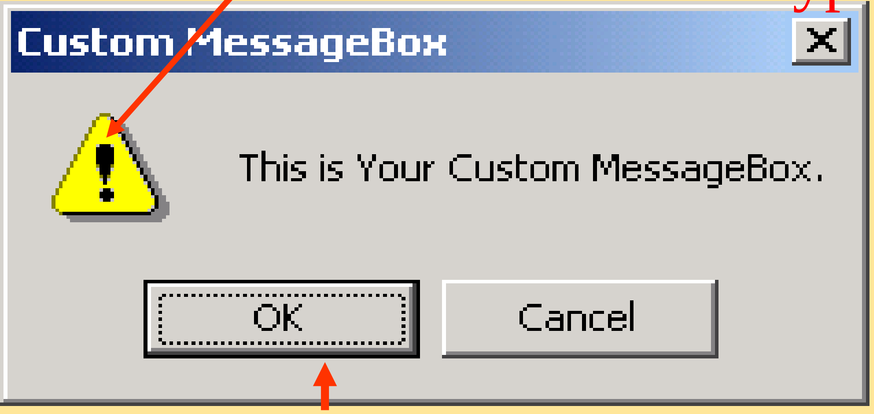
```
[STAThread]
static void Main() {
   Application.Run( new RadioButtonsTest() );
}
private void buttonType_CheckedChanged(
object sender, System.EventArgs e ) {
   if ( sender == okButton )
      buttonType = MessageBoxButtons.OK;
   else if ( sender == okCancelButton )
      buttonType = MessageBoxButtons.OKCancel;
   else if ( sender == abortRetryIgnoreButton )
      buttonType = MessageBoxButtons.AbortRetryIgnore;
   else if ( sender == yesNoCancelButton )
      buttonType = MessageBoxButtons.YesNoCancel;
   else if ( sender == yesNoButton )
      buttonType = MessageBoxButtons.YesNo;
   else
      buttonType = MessageBoxButtons.RetryCancel;
}
```

Visually new this event handler and set checkedChange for all buttons to this new handler (setting codes in initializeComponent

```
      private void iconType_CheckedChanged(
        object sender, System.EventArgs e ) {
        if ( sender == errorButton )
           iconType = MessageBoxIcon.Error;
        else if ( sender == exclamationButton )
           iconType = MessageBoxIcon.Exclamation
        else if ( sender == informationButton )
           iconType = MessageBoxIcon.Information;
        else // only one option left--display question mark
           iconType = MessageBoxIcon.Question;
      }
      protected void displayButton_Click(
       object sender, System.EventArgs e ) {
         DialogResult result = MessageBox.Show( "This is Your Custom
           MessageBox.", "Custom MessageBox",buttonType, iconType, 0, 0 );
```

Visually new this event handler and set checkedChange for all buttons to this new handler

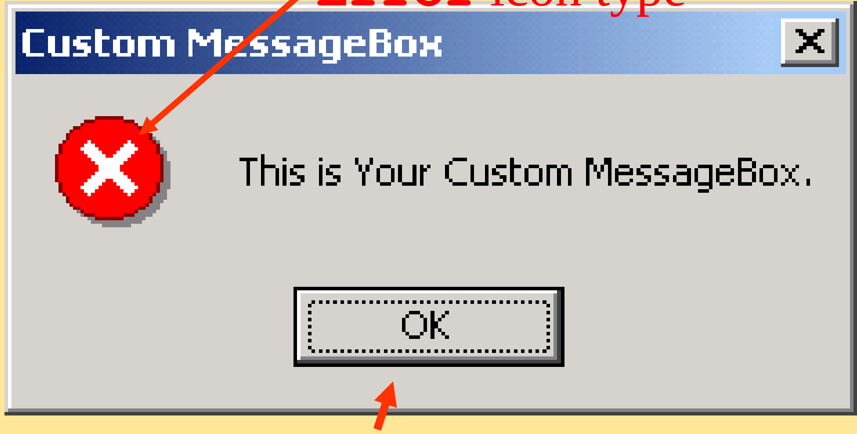
Exclamation icon type


Custom MessageBox

This is Your Custom MessageBox.

OKCancel button type

Error icon type

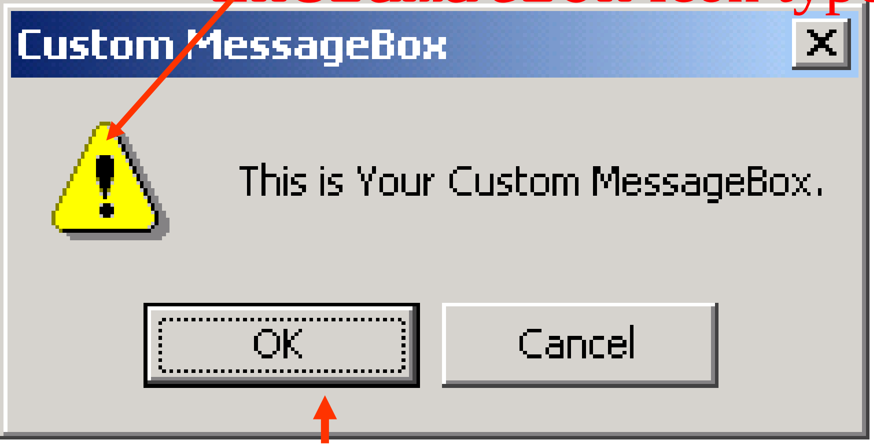

Custom MessageBox

This is Your Custom MessageBox.

OK button type

```
        switch ( result ) {
          case DialogResult.OK:
            displayLabel.Text = "OK was pressed.";
            break;
          case DialogResult.Cancel:
            displayLabel.Text = "Cancel was pressed.";
            break;
          case DialogResult.Abort:
            displayLabel.Text = "Abort was pressed.";
            break;
          case DialogResult.Retry:
            displayLabel.Text = "Retry was pressed.";
            break;
          case DialogResult.Ignore:
            displayLabel.Text = "Ignore was pressed.";
            break;
          case DialogResult.Yes:
            displayLabel.Text = "Yes was pressed.";
            break;
          case DialogResult.No:
            displayLabel.Text = "No was pressed.";
            break;
        }
    }
}
```

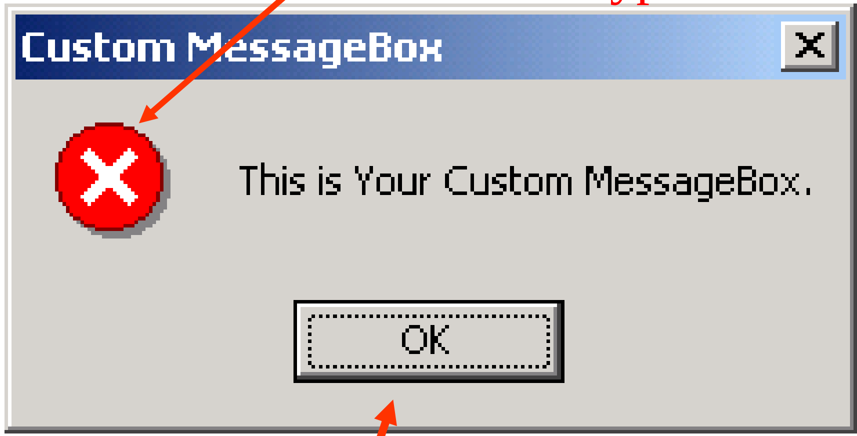
Exclamation icon type


Custom MessageBox

This is Your Custom MessageBox.

OKCancel button type

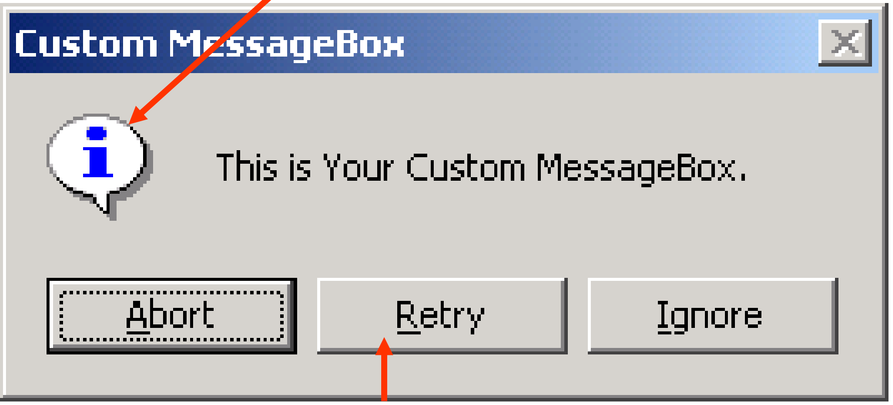
Error icon type


Custom MessageBox

This is Your Custom MessageBox.

OK button type

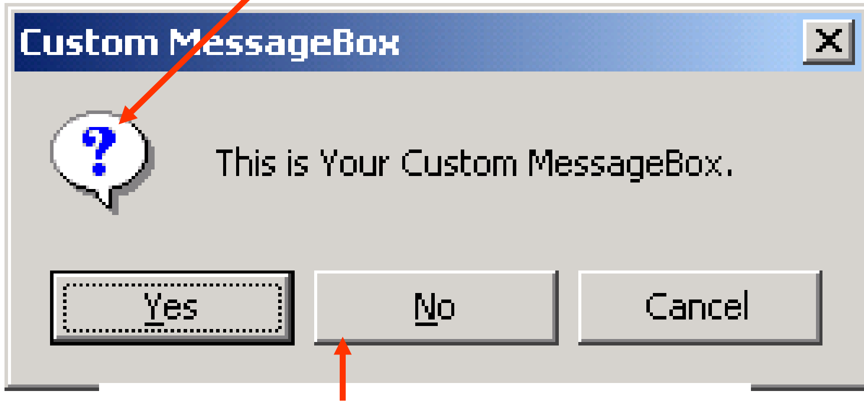
Information icon type


Custom MessageBox

This is Your Custom MessageBox.

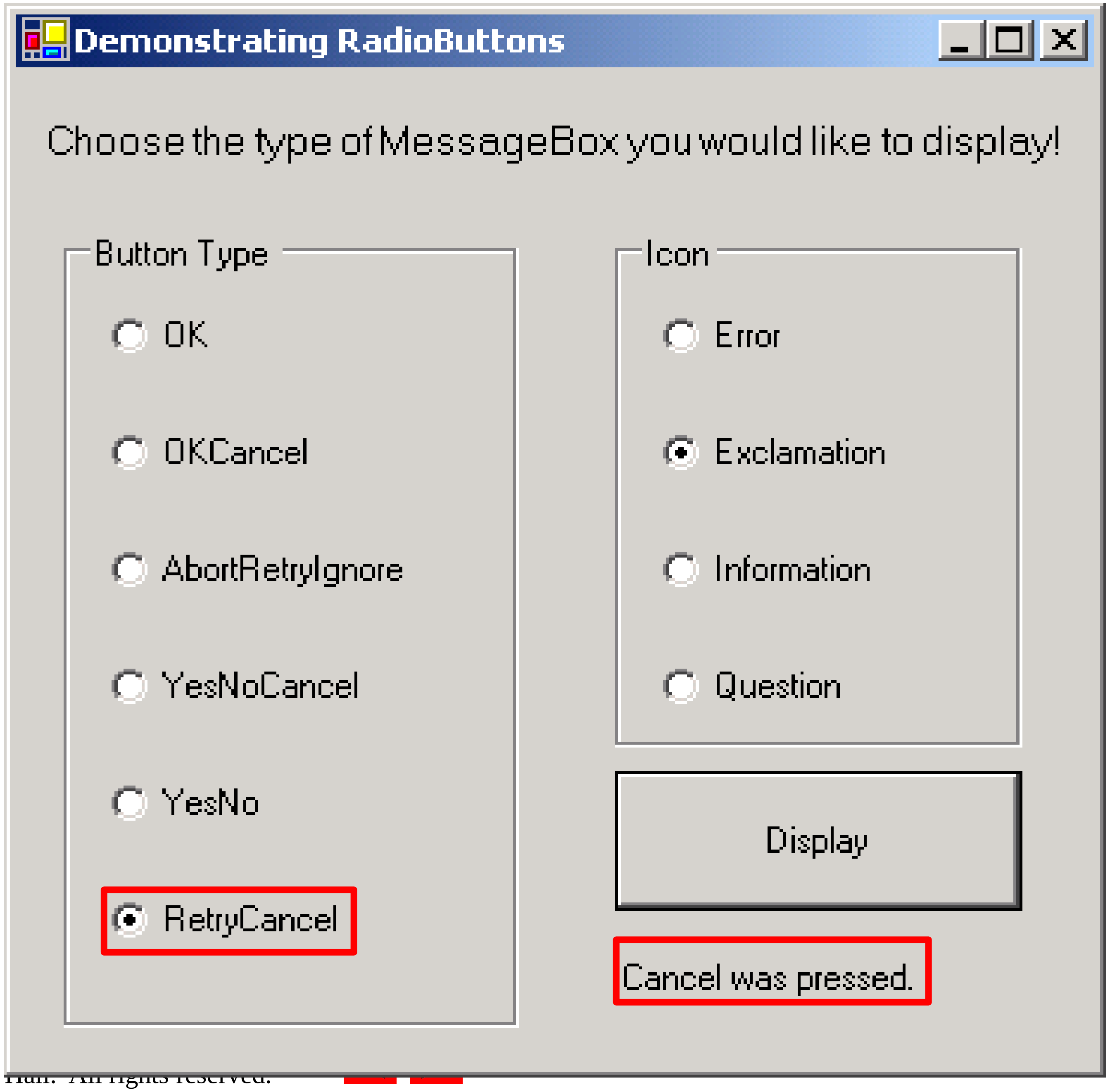
AbortRetryIgnore button type

Question icon type


Custom MessageBox

This is Your Custom MessageBox.

YesNoCancel button type



Demonstrating RadioButtons

Choose the type of MessageBox you would like to display!

Button Type

- ( ) OK
- ( ) OKCancel
- ( ) AbortRetryIgnore
- ( ) YesNoCancel
- ( ) YesNo
- (●) RetryCancel

Icon

- ( ) Error
- (●) Exclamation
- ( ) Information
- ( ) Question

Display

Cancel was pressed.

12.8 PictureBoxes

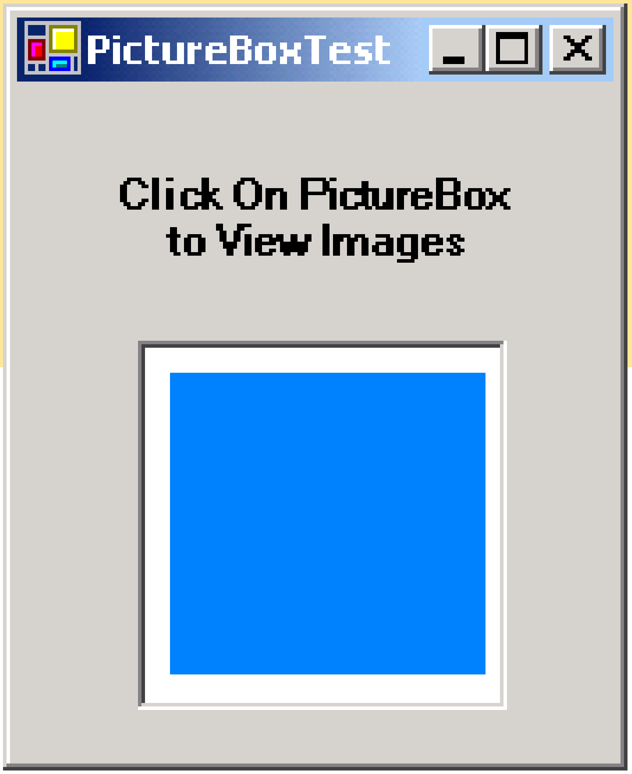
- Class PictureBox
  - Displays an image
    - Image set by object of class Image.
      - The Image property sets the Image object to use
      - SizeMode property sets how the image is displayed

`PictureBox` properties and events	Description / Delegate and Event Arguments
Common Properties	
`Image`	Image to display in the **`PictureBox`**.
`SizeMode`	Enumeration that controls image sizing and positioning. Values **`Normal`** (default), **`StretchImage`**, **`AutoSize`** and **`CenterImage`**. **`Normal`** puts image in top-left corner of **`PictureBox`** and **`CenterImage`** puts image in middle. (Both cut off image if too large.) **`StretchImage`** resizes image to fit in **`PictureBox`**. **`AutoSize`** resizes **`PictureBox`** to hold image.
Common Events	*(Delegate* **`EventHandler`***, event arguments* **`EventArgs`***)*
`Click`	Raised when user clicks the control. Default event when this control is double clicked in the designer.

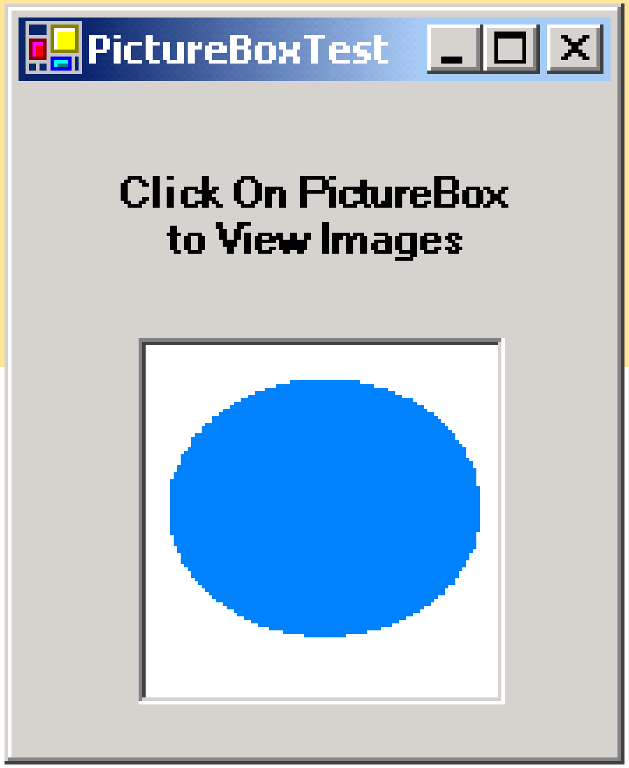
```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;
 public class PictureBoxTest : System.Windows.Forms.Form {
    private System.Windows.Forms.PictureBox imagePictureBox;
    private System.Windows.Forms.Label promptLabel;
    private int imageNum = -1;
    [STAThread]
    static void Main() {
        Application.Run( new PictureBoxTest() );
    }
    private void imagePictureBox_Click(
        object sender, System.EventArgs e ) {
        imageNum = ( imageNum + 1 ) % 3;
```

imageNum in {0, 1, 2}

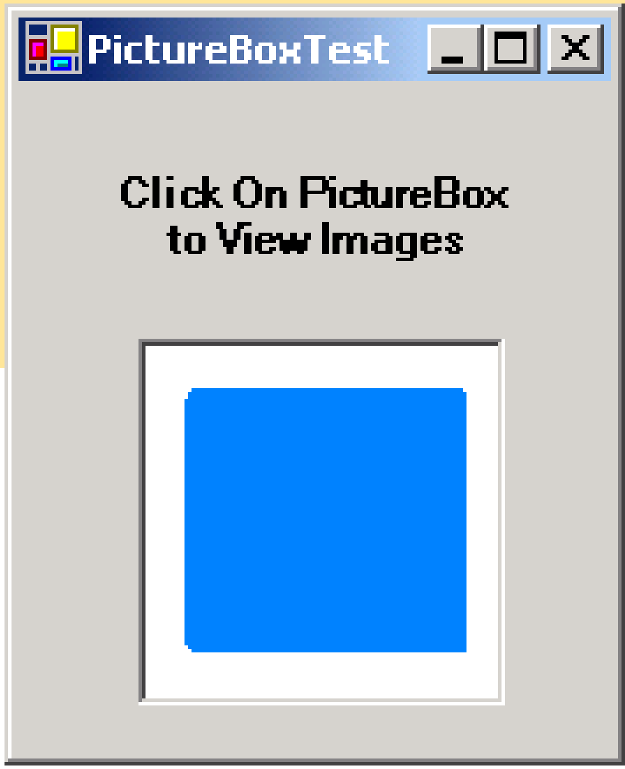
```
         imagePictureBox.Image = Image.FromFile(
          Directory.GetCurrentDirectory()+ "\\images\\image" +
          imageNum + ".bmp" );
      }
   }
```


PictureBoxTest

Click On PictureBox to View Images


PictureBoxTest

Click On PictureBox to View Images


PictureBoxTest

Click On PictureBox to View Images

Mouse Events, Delegates and Event Arguments

Mouse Events (Delegate **`EventHandler`**, *event arguments* **`EventArgs`***)*	
`MouseEnter`	Raised if the mouse cursor enters the area of the control.
`MouseLeave`	Raised if the mouse cursor leaves the area of the control.
Mouse Events (Delegate **`MouseEventHandler`**, *event arguments* **`MouseEventArgs`***)*	
`MouseDown`	Raised if the mouse button is pressed while its cursor is over the area of the control.
`MouseHover`	Raised if the mouse cursor hovers over the area of the control.
`MouseMove`	Raised if the mouse cursor is moved while in the area of the control.
`MouseUp`	Raised if the mouse button is released when the cursor is over the area of the control.
Class **`MouseEventArgs`** *Properties*	
`Button`	Mouse button that was pressed (**`left`**, **`right`**, **`middle`** or **`none`**).
`Clicks`	The number of times the mouse button was clicked.
`X`	The *x*-coordinate of the event, relative to the control.
`Y`	The *y*-coordinate of the event, relative to the control.

12.9 Mouse Event Handling

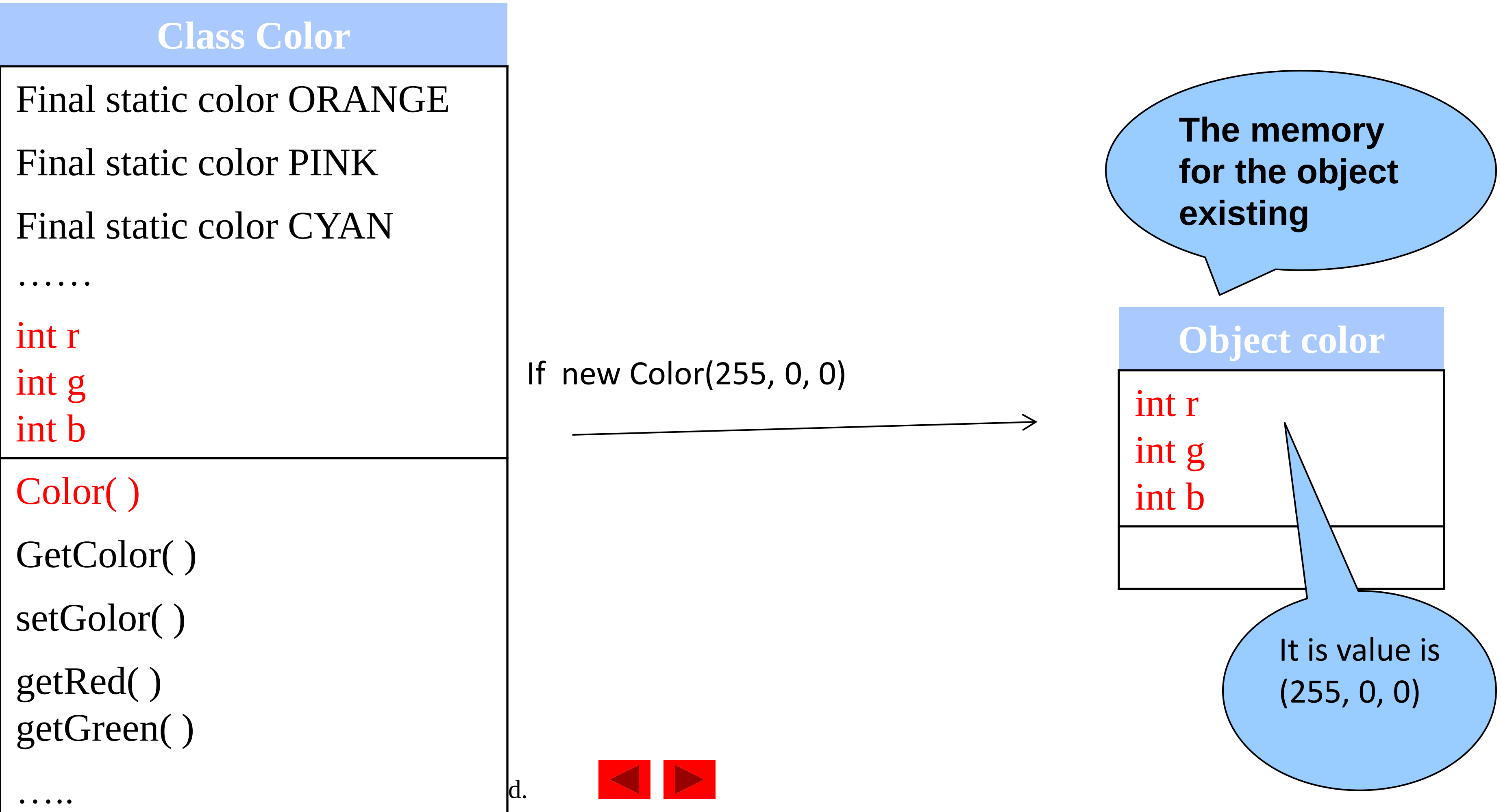
- Class MouseEventArgs
  - Contain coordinates of the mouse pointer
  - The mouse pressed
  - Number of clicks
  - Number of notches the wheel turned
  - Passing mouse event
  - Mouse event-handling methods take an object and MouseEventArgs object as argument
- The Click event uses delegate EventHandler and event arguments EventArgs

Java's constant variables of `Color class`

`Color` constant	Color	RGB value
`public final static Color ORANGE`	orange	255, 200, 0
`public final static Color PINK`	pink	255, 175, 175
`public final static Color CYAN`	cyan	0, 255, 255
`public final static Color MAGENTA`	magenta	255, 0, 255
`public final static Color YELLOW`	yellow	255, 255, 0
`public final static Color BLACK`	black	0, 0, 0
`public final static Color WHITE`	white	255, 255, 255
`public final static Color GRAY`	gray	128, 128, 128
`public final static Color LIGHT_GRAY`	light gray	192, 192, 192
`public final static Color DARK_GRAY`	dark gray	64, 64, 64
`public final static Color RED`	red	255, 0, 0
`public final static Color GREEN`	green	0, 255, 0
`public final static Color BLUE`	blue	0, 0, 255

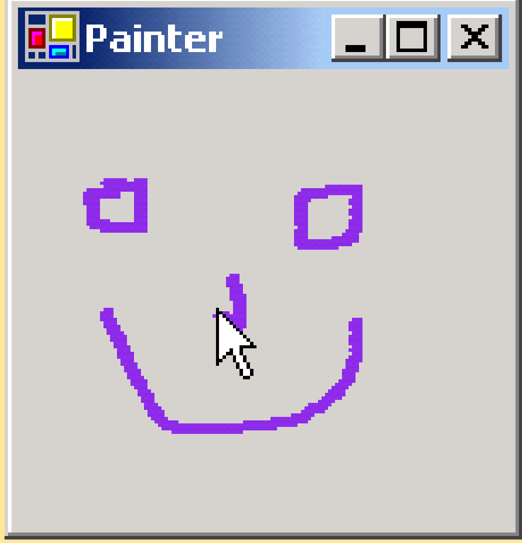
Java's Color Control

- Class `Color`
  - Defines methods and constants for manipulating colors
  - Colors are created from red, green and blue components
    - You can tune its RGB values



d.

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
public class Painter : System.Windows.Forms.Form {
    bool shouldPaint = false;
    [STAThread]
    static void Main() {
        Application.Run( new Painter() );
    }
    private void Painter_MouseDown(
        object sender, System.Windows.Forms.MouseEventArgs e ) {
        shouldPaint = true;
    }
    private void Painter_MouseUp(
        object sender, System.Windows.Forms.MouseEventArgs e ) {
        shouldPaint = false;
    }
```

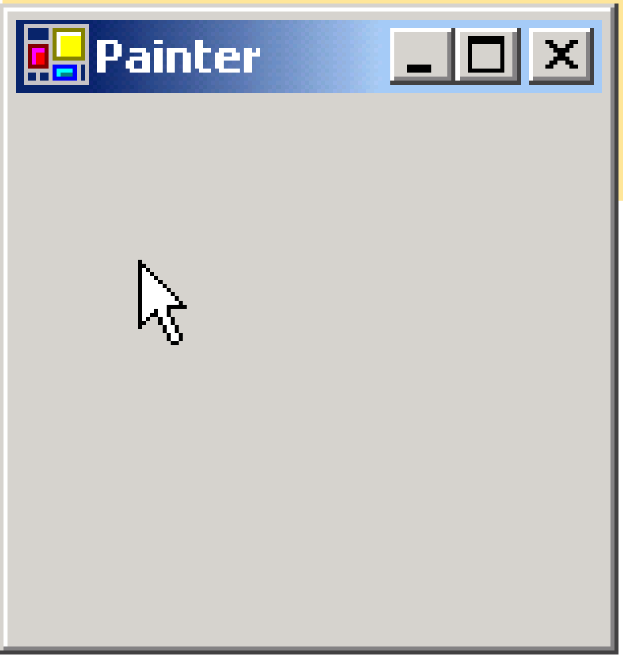


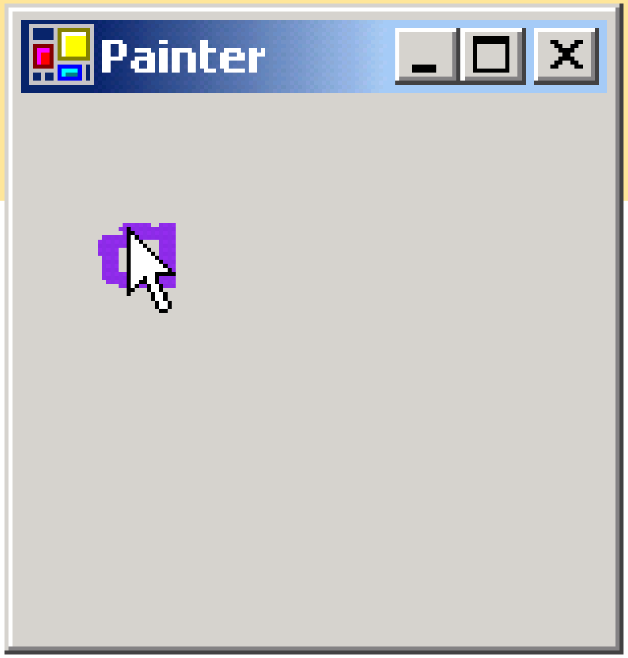
```
protected void Painter_MouseMove(
      object sender, System.Windows.Forms.MouseEventArgs e ) {
    if ( shouldPaint ) {
      Graphics graphics = CreateGraphics();
      graphics.FillEllipse(new SolidBrush(Color.BlueViolet ), e.X, e.Y, 4, 4);
    }
  }
}
```

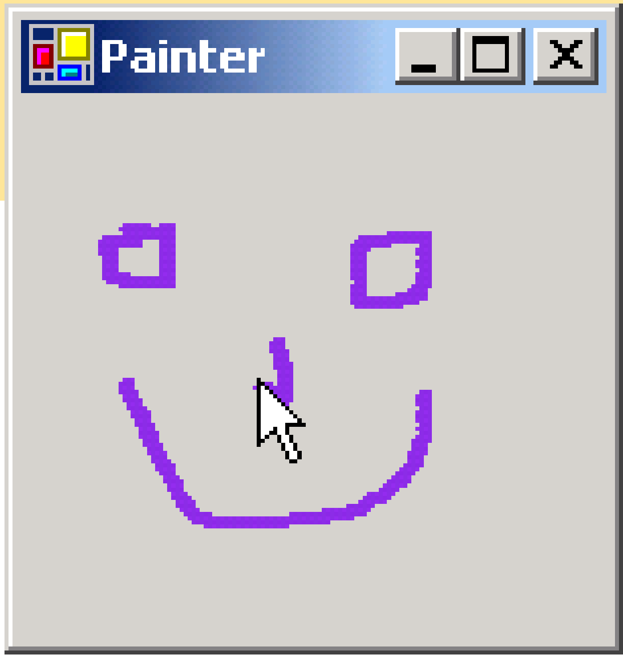
size

Position

CreateGraphics() is a method of Control, it returns a Graphics object for the control







Keyboard Events, Delegates and Event Arguments	
Key Events (Delegate **KeyEventHandler***, event arguments* **KeyEventArgs***)*	
KeyDown	Raised when key is initially pushed down.
KeyUp	Raised when key is released.
Key Events (Delegate **KeyPressEventHandler***, event arguments* **KeyPressEventArgs***)*	
KeyPress	Raised when key is pressed. Occurs repeatedly while key is held down, at a rate specified by the operating system.
Class **KeyPressEventArgs** *Properties*	
KeyChar	Returns the ASCII character for the key pressed.
Handled	Whether the **KeyPress** event was handled.
Class **KeyEventArgs** *Properties*	
Alt	Indicates whether the *Alt* key was pressed.
Control	Indicates whether the *Control* key was pressed.
Shift	Indicates whether the *Shift* key was pressed.
Handled	Whether the event was handled.
KeyCode	Returns the key code for the key, as a **Keys** enumeration. This does not include modifier key information. Used to test for a specific key.
KeyData	Returns the key code as a **Keys** enumeration, combined with modifier information. Used to determine all information about the key pressed.
KeyValue	Returns the key code as an **int**, rather than as a **Keys** enumeration. Used to obtain a numeric representation of the key pressed.
Modifiers	Returns a **Keys** enumeration for any modifier keys pressed (*Alt*, *Control* and *Shift*). Used to determine modifier key information only.

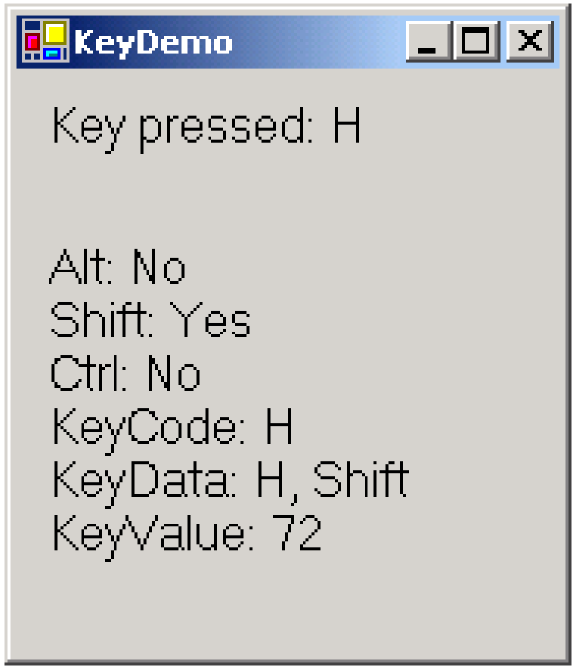
12.10 Keyboard Event Handling

- Key events
  - Control that inherits from System.Windows.Forms.Control
  - Delegate KeyPressEventHandler
    - Event argument KeyPressEventArgs
    - KeyPress
      - ASCII character pressed
      - No modifier keys
  - Delegate KeyEventHandler
    - Event argument KeyEventArgs
    - KeyUp or KeyDown
      - Special modifier keys
    - Key enumeration value

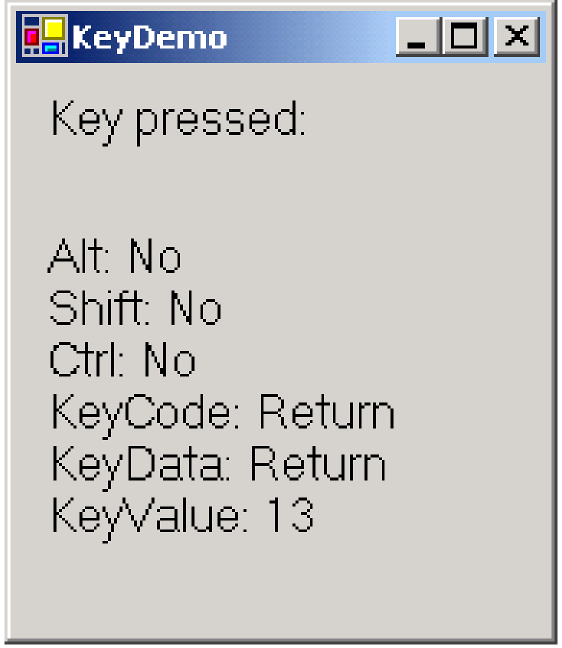
```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
public class KeyDemo : System.Windows.Forms.Form {
   private System.Windows.Forms.Label charLabel;
   private System.Windows.Forms.Label keyInfoLabel;
   private System.ComponentModel.Container components = null;
   [STAThread]
   static void Main() {
      Application.Run( new KeyDemo() );
   }
   protected void KeyDemo_KeyPress(
   object sender, System.Windows.Forms.KeyPressEventArgs e) {
      charLabel.Text = "Key pressed: " + e.KeyChar;
   }
```

```
    private void KeyDemo_KeyDown(
     object sender, System.Windows.Forms.KeyEventArgs e ) {
        keyInfoLabel.Text =
           "Alt: " + ( e.Alt ? "Yes" : "No") + '\n' +
           "Shift: " + ( e.Shift ? "Yes" : "No" ) + '\n' +
           "Ctrl: " + ( e.Control ? "Yes" : "No" ) + '\n' +
           "KeyCode: " + e.KeyCode + '\n' +
           "KeyData: " + e.KeyData + '\n' +
           "KeyValue: " + e.KeyValue;
     }
     private void KeyDemo_KeyUp(
     object sender, System.Windows.Forms.KeyEventArgs e ){
        keyInfoLabel.Text = "";
        charLabel.Text = "";
     }
```

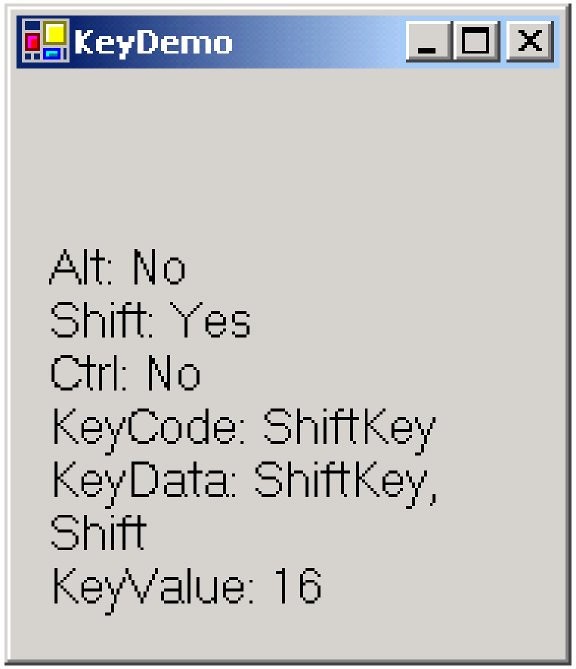
KeyDemo.cs
Program Output



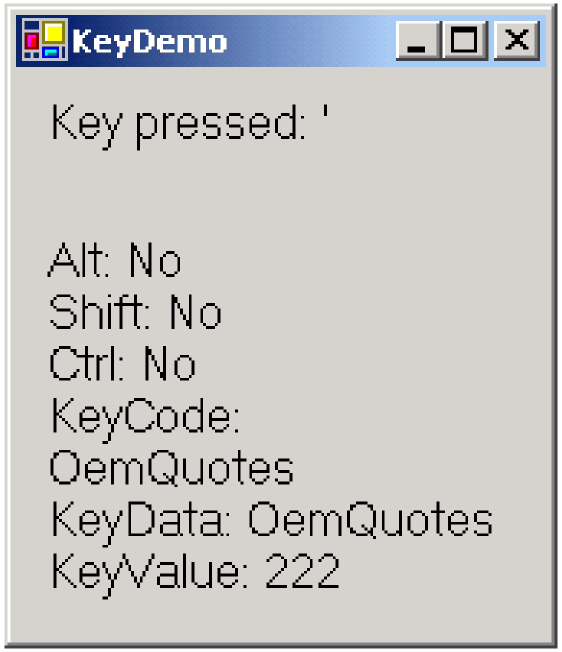
```
Key pressed: H

Alt: No
Shift: Yes
Ctrl: No
KeyCode: H
KeyData: H, Shift
KeyValue: 72
```



```
Key pressed:

Alt: No
Shift: No
Ctrl: No
KeyCode: Return
KeyData: Return
KeyValue: 13
```



```
Alt: No
Shift: Yes
Ctrl: No
KeyCode: ShiftKey
KeyData: ShiftKey,
Shift
KeyValue: 16
```



```
Key pressed: '

Alt: No
Shift: No
Ctrl: No
KeyCode:
OemQuotes
KeyData: OemQuotes
KeyValue: 222
```